ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Ανάπτυξη Πλατφόρμας Μοντελοκεντρικής Αρχιτεκτονικής για Κατανεμημένα Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννης Ε. Φουκαράκης

Αθήνα, Ιανουάριος 2008

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Ανάπτυξη Πλατφόρμας Μοντελοκεντρικής Αρχιτεκτονικής για Κατανεμημένα Συστήματα

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Ιωάννης Ε. Φουκαράκης

Συμβουλευτική Επιτροπή : Δήμητρα Ι. Κακλαμάνη
Ιάκωβος Στ. Βενιέρης
Νικόλαος Κ. Ουζούνογλου

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 28η Ιανουαρίου 2008.

.....	.....	.....
Δήμητρα Κακλαμάνη	Ιάκωβος Βενιέρης	Νικόλαος Ουζούνογλου
Αν. Καθ. ΕΜΠ, ΣΗΜΜΥ	Καθ. ΕΜΠ, ΣΗΜΜΥ	Καθ. ΕΜΠ, ΣΗΜΜΥ
.....	.....	.....
Στέφανος Κόλλιας	Γεώργιος Στασινόπουλος	Θεοδώρα Βαρβαρίγου
Καθ. ΕΜΠ, ΣΗΜΜΥ	Καθ. ΕΜΠ, ΣΗΜΜΥ	Καθ. ΕΜΠ, ΣΗΜΜΥ
	.....	
	Χρήστος Κακλαμάνης	
	Καθ. Πανεπ. Πατρών, ΤΜΗΥΠ	

Αθήνα, Ιανουάριος 2008

.....

Ιωάννης Ε. Φουκαράκης

Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

ΠΕΡΙΛΗΨΗ

Η εμφάνιση νέων τεχνολογιών που αξιοποιούν ολοένα και περισσότερους πόρους στο χώρο των κατανεμημένων συστημάτων είναι ιδιαίτερα δελεαστική για εφαρμογές απαιτητικές σε υπολογιστική ισχύ, μνήμη και αποθηκευτικό χώρο. Ο σκοπός αυτής της εργασίας ήταν η δημιουργία μιας πλατφόρμας που θα επιτρέψει τη γρήγορη και εύκολη μεταφορά εφαρμογών σε διαφορετικά κατανεμημένα περιβάλλοντα.

Στο πρώτο στάδιο της εργασίας αναπτύχτηκε μία πλατφόρμα κινητών πρακτόρων η οποία μπορεί να ενσωματωθεί σε εξυπηρετητές ιστού. Η πλατφόρμα αυτή προσφέρει τις βασικές λειτουργίες μιας πλατφόρμας κινητών πρακτόρων όπως η δυνατότητα φιλοξενίας αυτόνομων κινητών πρακτόρων, οι οποίοι έχουν δυνατότητα μετακίνησης, δυνατότητα αναζήτησης και επικοινωνίας των κινητών αντιπροσώπων, καθώς και ανάπτυξη εργαλείων διαχείρισης της πλατφόρμας. Πρωτεύον συστατικό της πλατφόρμας είναι η τεχνολογία SOAP, η οποία επιτρέπει την επικοινωνία ανάμεσα σε απομακρυσμένους υπολογιστές, πάνω από το πρωτόκολλο HTTP.

Ο χρήστης που επιθυμεί να χρησιμοποιήσει την πλατφόρμα, έχει τη δυνατότητα να αναπτύξει ολοκληρωμένους κινητούς πράκτορες, οι οποίοι μπορούν να τον βοηθήσουν σε εφαρμογές ηλεκτρονικού εμπορίου, διαχείρισης απομακρυσμένων υπολογιστών, επίλυση κατανεμημένων προβλημάτων κλπ.

Εν συνεχεία μελετήθηκε η τεχνολογία του Υπολογιστικού Πλέγματος (GRID) μέσω των υποδομών που προσφέρει το EGEE. Στα πλαίσια αυτής της έρευνας αναπτύχθηκε πρότυπη δικτυακή πλεγματική πύλη η οποία αξιοποιεί τους πόρους του Πλέγματος προκειμένου να επιλύσει ένα παραμετρικό πρόβλημα διπλώματος πρωτεϊνών για ένα μεγάλο όγκο δεδομένων, ενώ ταυτόχρονα χρησιμοποιεί μοντέρνες τεχνολογίες διαδικτύου που επιτρέπουν καλύτερη και πληρέστερη οργάνωση ομάδων χρηστών γύρω από τις εφαρμογές.

Η ενασχόληση με τα δύο αυτά διαφορετικά κατανεμημένα συστήματα ανέδειξε το πρόβλημα ότι η διαδικασία μεταφοράς ήδη υπαρχουσών εφαρμογών κατανεμημένα στα συστήματα, αλλά και ανάμεσα σε διαφορετικά κατανεμημένα συστήματα είναι μια επίπονη διαδικασία, παρόλο το ότι υπάρχουν ικανά εργαλεία διαθέσιμα. Προκειμένου να διευκολυνθεί η διαδικασία μεταφοράς μελετήθηκαν οι βασικές αρχές της μοντελοκεντρικής αρχιτεκτονικής. Χρησιμοποιώντας παραδοσιακές τεχνικές σχεδιασμού λογισμικού έγινε δυνατή η δημιουργία του μεγαλύτερου μέρους του κώδικα που απαιτείται για την τελική κατανεμημένη εφαρμογή.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Τεχνολογίες κατανεμημένων αντικειμένων, παράλληλα κατανεμημένα συστήματα, μεσισμικό, κινητοί πράκτορες, Υπηρεσίες Ιστού, SOAP, κινητός κώδικας, XML, Πλέγμα Μοντελοκεντρική αρχιτεκτονική, UML, σχεδιαστικά πρότυπα.

ABSTRACT

During the last years has been a great development of new technologies that utilize more resources in the area of distributed systems. The large amount of resources available nowadays is attractive to the users of existing applications that require large amount of computational power, memory and storage area. The goal of the work presented in this thesis is to propose a platform that enables fast and easy porting of existing applications to different distributed environments.

The first step of the research was to explore the capabilities of mobile agent technology via the integration of a mobile agent platform within a web server. Such a capability is very important not only for its applications, but also because of the computational power that we can obtain from the existing infrastructure of the Web. This power can be manipulated in order to help us solve more complex problems that require a handful of resources.

The existing form of the project offers the basic functionalities of a mobile agent platform such as the ability to host agents that can move to a different location, search and communication interfaces and management tools. The primary component of the platform is the SOAP technology, which enables the communication of remote hosts over HTTP.

Every user willing to facilitate the platform is provided with the tools to develop full-scale mobile agents that can help him with applications such as e-commerce, remote computer management, computational problems etc.

The increasing hype of the Grid technology. Within the scope of this research, a Grid portal was developed. The Grid portal utilizes the resources available at the Grid in order to solve a large parametric problem of protein folding for a large amount of input data, while it supports improved communication between the users of the application by providing new internet technologies.

Although these distributed systems offer tools eligible for the development of different applications, the process of porting existing applications to or between them is not easy. In order to ease the porting process, the main principles of model driven architecture and agile development were investigated. By using traditional design patterns and technologies the generation of an extensive part of the code of the final distributed application was achieved.

KEYWORDS

Distributed object technologies, parallel distributed computing, middleware, mobile agents, Web Services, SOAP, mobile code, Grid, Model-driver architecture, UML, design patterns

Η παρούσα διδακτορική διατριβή εκπονήθηκε στο Εργαστήριο Μικροκυμάτων και Οπτικών Ινών της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Θα ήθελα να εκφράσω τις θερμές ευχαριστίες μου στην επιβλέπουσα καθηγήτρια κ. Δήμητρα Κακλαμάνη για τις πολύτιμες συμβουλές της, την καθοδήγησή της και την αμέριστη συμπαράσταση που μου έδειξε καθ' όλη τη διάρκεια της συνεργασίας μας.

Ευχαριστώ επίσης τον καθηγητή κ. Ιάκωβο Βενιέρη για την άριστη συνεργασία μας τόσο στα πλαίσια της διδακτορικής διατριβής όσο και σε ερευνητικές δραστηριότητες, καθώς οι προτάσεις του οδήγησαν την έρευνά μου σε νέα πεδία.

Ευχαριστώ θερμά το διευθυντή του εργαστηρίου καθηγητή κ. Νικόλαο Ουζούνογλου για την εμπιστοσύνη που μου επέδειξε στη διάρκεια της πορείας μου ως υποψήφιος διδάκτορας.

Θερμές ευχαριστίες οφείλω και στον καθηγητή κ. Ηλία Ηλιόπουλο του Γεωπονικού Πανεπιστημίου Αθηνών για την εξαίρετη συνεργασία μας.

Θα ήθελα ιδιαίτερα να ευχαριστήσω τους συνάδελφούς μου και φίλους Δρ. Αντώνη Κωσταρίδη, Δρ. Δημήτρη Λυμπερόπουλο, Δρ. Χρήστο Μπίνιαρη, Δρ. Βαγγέλη Αγγελόπουλο, καθ. Γιώργο Πρεζεράκο, Δρ. Νίκο Τσελίκα, Δρ. Σοφία Καπελάκη, Κώστα Παπαδόπουλο, Θοδωρή Αθαναηλέα, Δήμητρα Ζαρμπούτη, Ξένια Παπαδομιχελάκη, Αθηνά Οικονόμου, Παναγιώτη Γκόνη, Δημήτρη Κατέρο, Χρήστο Κατσιγιάννη, Δημήτρη Τσιλιμαντό, Γεωργία Καπιτσάκη, και Γιώργο Λιουδάκη για την εξαίρετη συνεργασία μας τα χρόνια που βρισκόμασταν στο ίδιο εργαστήριο.

Επίσης ένα ιδιαίτερο ευχαριστώ στους φίλους και ανθρώπους που με στήριξαν σε δύσκολες στιγμές κατά τη διάρκεια της εκπόνησης της διατριβής

Τέλος, ιδιαίτερα θέλω να ευχαριστήσω την οικογένειά μου η οποία με στήριξε ηθικά και μου πρόσφερε τα απαραίτητα εφόδια ώστε να μπορέσω να ολοκληρώσω τις σπουδές μου.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1 ΓΕΝΙΚΗ ΕΠΙΣΚΟΠΗΣΗ

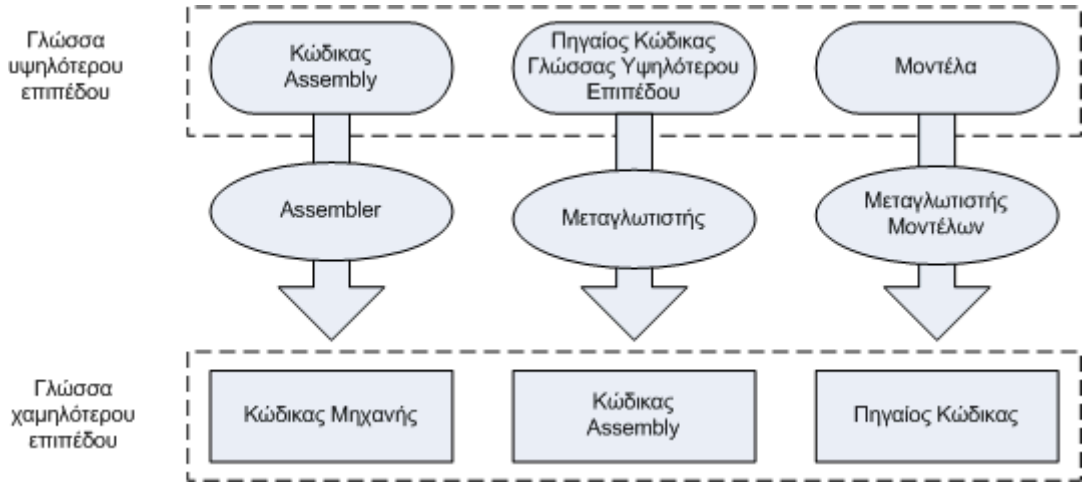
Παρατηρώντας την ιστορία της πληροφορικής και των υπολογιστών, διαπιστώνουμε ότι η τάση που επικρατεί στο σχεδιασμό των συστημάτων και των προγραμμάτων είναι να γίνεται ολοένα και πιο αφαιρετικός. Στους πρώτους υπολογιστές, ανάλογα με το πώς συνδέονταν τα καλώδια προέκυπταν τα προγράμματα. Στη συνέχεια, αλλάζοντας τις θέσεις κάποιων διακοπτών άλλαζε και το πρόγραμμα. Η πρώτη μεγάλη αλλαγή έγινε με τη γλώσσα assembly, όπου ο προγραμματιστής μπορούσε να γράψει προγράμματα σε γλώσσα μηχανής και η μηχανή να ακολουθήσει τις εκάστοτε οδηγίες που είχε.

Το επόμενο μεγάλο βήμα ήταν οι γλώσσες προγραμματισμού, που επέτρεψαν την ανάπτυξη προγραμμάτων σε μια γλώσσα ευκολότερα κατανοητή από τον άνθρωπο. Καθορίστηκαν στάνταρτ για τις γλώσσες (όπως π.χ. η COBOL και η C), με αποτέλεσμα να είναι δυνατή η χρησιμοποίησή τους σε πολλές διαφορετικές πλατφόρμες, ενώ η ανάπτυξη, η κατανόηση και η συντήρηση των προγραμμάτων έγιναν πιο εύκολες. Με τον καιρό αναπτύχθηκαν νέες γλώσσες προσφέροντας νέες δυνατότητες και διαφορετική προσέγγιση στο τρόπο που γράφεται ο κώδικας. Έτσι εμφανίστηκαν οι συναρτησιακές γλώσσες προγραμματισμού, και ακολούθησαν οι αντικειμενοστραφείς γλώσσες, οι οποίες επέτρεψαν μέσα στην έννοια της κλάσης να ενσωματωθούν τόσο τα δεδομένα, όσο και οι διαδικασίες που ενεργούν πάνω σε αυτά.

Αυτό που προσέφεραν οι νέες γλώσσες προγραμματισμού σε σχέση με τις παλαιότερες είναι ένα υψηλότερο επίπεδο αφαιρετικότητας. Κάθε φορά που εμφανιζόταν μια νέα γλώσσα υψηλότερου επιπέδου έπρεπε να γίνει κάποια αντιστοίχηση σε γλώσσες χαμηλότερου επιπέδου. Για παράδειγμα, ο κώδικας της C μετατρέπεται σε γλώσσα μηχανής και μετά στις λειτουργίες του υπολογιστικού συστήματος. Οι κανόνες για την αντιστοίχιση αυτή δεν ορίζονται από την αρχή. Το νέο επίπεδο αφαιρετικότητας (π.χ. ο αντικειμενοστραφής προγραμματισμός) αρχικά υπάρχει σαν έννοια, και σιγά – σιγά αναπτύσσονταν τα εργαλεία που βοηθούσαν στην αυτοματοποίηση αυτής της διαδικασίας. Για παράδειγμα, τα πρώτα προγράμματα assembly μετατρεπόνταν σε bits με το χέρι, μέχρι που εμφανίστηκαν οι πρώτοι assemblers.

Η διαδοχική αυτή προσθήκη επιπλέον αφαιρετικότητας στο επίπεδο στο οποίο αναπτύσσεται ο κώδικας συνοδεύτηκε με την ανάπτυξη εργαλείων που αντιστοιχίζουν λειτουργίες κάθε επιπέδου σε λειτουργίες του αμέσως χαμηλότερου επιπέδου. Έτσι, οι προγραμματιστές σήμερα γράφουν κώδικα σε γλώσσες υψηλού επιπέδου ο οποίος μετατρέπεται σε γλώσσα χαμηλότερου επιπέδου αυτόματα, όπως παλιότερα ο κώδικας σε

γλώσσα χαμηλότερου επιπέδου μετατρεπόταν αυτόματα σε γλώσσα μηχανής και ακόμα πιο παλιά ο assembler μετέτρεπε τον κώδικα σε γλώσσα μηχανής. Αντίστοιχα, στο μέλλον ο κώδικας σε μια γλώσσα υψηλότερου επιπέδου θα μετατρέπεται σε κώδικα γλώσσας υψηλού επιπέδου.



Εικόνα 1.1: Προσθήκη αφαιρετικότητας

Από τα παραπάνω φαίνεται η διαδικασία που ακολουθείται τα τελευταία χρόνια. Μόλις μια τεχνολογία γίνει κατανοητή και αρχίσουν να εφαρμόζονται κάποιοι κανόνες, αυτοί επισημοποιούνται και έτσι προκύπτει μια γλώσσα υψηλότερου επιπέδου. Μέσω των κανόνων, μπορεί να γίνει η αντιστοίχηση από τη νέα γλώσσα στην παλιά. Η διαδικασία αυτή επαναλαμβάνεται συνεχώς.

Μια λύση που φαίνεται πολύ πιθανή να είναι το επόμενο επίπεδο αφαιρετικότητας είναι η ανάπτυξη εφαρμογών βασισμένη στο μοντέλο. Σε αυτή την περίπτωση η ανάπτυξη του μοντέλου είναι ανεξάρτητη από την πλατφόρμα λογισμικού που θα χρησιμοποιηθεί.

Η ανεξαρτησία από τη πλατφόρμα λογισμικού μπορεί να παραλληλιστεί με την ανεξαρτησία από την πλατφόρμα υλικού. Όπως η C και η Java μπορούν να χρησιμοποιηθούν σε διαφορετικά υπολογιστικά συστήματα για την ανάπτυξη εφαρμογών, έτσι και η ανάπτυξη εφαρμογών βάσει του μοντέλου θα επιτρέψει το σχεδιασμό προδιαγραφών που θα μπορούν να υλοποιηθούν σε διάφορες πλατφόρμες και αρχιτεκτονικές λογισμικού, χωρίς να απαιτείται αλλαγή στο μοντέλο. Έτσι ένα μοντέλο θα μπορεί π.χ. να αντιστοιχηθεί σε μια εφαρμογή Web Services, σε μια εφαρμογή peer-2-peer ή σε μια εφαρμογή κινητών πρακτόρων.

Η προσπάθεια της αντιμετώπισης ολοένα και πιο απαιτητικών υπολογιστικών προβλημάτων, καθώς και της εκτέλεσής τους σε όσο το δυνατόν λιγότερο χρόνο, αποτελεί θεμελιώδη ερευνητική δραστηριότητα στον τομέα της προσομοίωσης συστημάτων

μετάδοσης πληροφορίας. Με δεδομένη την ανάπτυξη του παγκόσμιου Ιστού και των συναφών τεχνολογιών, το Διαδίκτυο παρουσιάζεται ως μια πανίσχυρη κατανεμημένη υπολογιστική μηχανή με επεξεργαστική ισχύ και υπολογιστικούς πόρους που ξεπερνούν τον πιο ισχυρό παράλληλο υπερυπολογιστή. Η εκμετάλλευση των πόρων του Διαδικτύου για κατανεμημένη παράλληλη επεξεργασία απαιτεί κατάλληλες αρχιτεκτονικές λογισμικού και συναφείς τεχνολογίες ικανές να αντιμετωπίσουν μια πληθώρα ιδιαίτερων χαρακτηριστικών που η φύση του Διαδικτύου επιβάλλει.

Η τεχνολογία των κινητών πρακτόρων είναι ιδανική για την ανάπτυξη εφαρμογών που αναλαμβάνουν να διευκολύνουν την εργασία του χρήστη για τη δημιουργία κατανεμημένων προσομοιώσεων. Οι κινητοί πράκτορες είναι αυτόνομες υπολογιστικές οντότητες, που έχουν την ικανότητα να επιλύουν συνεργατικά ένα πρόβλημα, χρησιμοποιώντας ανταλλαγή μηνυμάτων και γνώσης. Επίσης, έχουν τη δυνατότητα μετανάστευσης μέσα στο κατανεμημένο περιβάλλον επεξεργασίας για την ολοκλήρωση συγκεκριμένων διεργασιών, ανάλογα με τη θέση τους στο περιβάλλον.

Η χρήση των κινητών πρακτόρων σαν ένα αφαιρετικό εργαλείο για το σχεδιασμό και τη κατασκευή συστημάτων προσφέρει πολλές δυνατότητες. Από τη μία πλευρά, οι κινητοί πράκτορες μπορούν να χρησιμοποιηθούν για την μοντελοποίηση πολύπλοκων συστημάτων ως ένα σύνολο από διακριτά ανεξάρτητα συστατικά. Παράλληλα, επιτρέπουν το συγκερασμό λειτουργικότητας από για διαφορετικά πεδία, όπως για παράδειγμα ο σχεδιασμός, η μάθηση και ο συντονισμός, σε ένα κοινό σώμα. Αυτός είναι και ο κύριος λόγος για τον οποίο οι προσπάθειες για την βελτίωση της τεχνολογίας των κινητών πρακτόρων δεν εστιάζονται σε έναν τομέα, αλλά σε διάφορες περιοχές, όπως αρχιτεκτονικές συστημάτων, τεχνολογία λογισμικού κλπ.

Ένα από τα σημαντικότερα μειονεκτήματα της τεχνολογίας των κινητών πρακτόρων είναι ότι αντιμετωπίζουν ένα ιδιαίτερα σημαντικό πρόβλημα όσο αφορά στην κλιμάκωση του μεγέθους των πόρων που χρησιμοποιούνται για την επίλυση των προβλημάτων. Λύση σε αυτό το πρόβλημα έρχεται να δώσει η τεχνολογία του Πλέγματος, η οποία προσφέρει τα κατάλληλα εργαλεία μέσω των οποίων χρήστες από όλο τον κόσμο μπορούν να υποβάλλουν εργασίες σε πόρους οι οποίοι βρίσκονται διάσπαρτοι σε όλο τον κόσμο.

Το κυριότερο πρόβλημα που εμφανίζεται κατά το σχεδιασμό εφαρμογών με χρήση κατανεμημένων τεχνολογιών είναι η πολυπλοκότητα που εμφανίζει η διαδικασία ανάπτυξης τους. Λύση σε αυτό το πρόβλημα φαίνεται να δίνει η μοντελοκεντρική αρχιτεκτονική. Η μοντελοκεντρική αρχιτεκτονική προσφέρει τα απαραίτητα εργαλεία ώστε να δημιουργείται η τελική εφαρμογή (ή τουλάχιστον ένα μεγάλο μέρος της) από το μοντέλο της εφαρμογής.

Έτσι, ο ρόλος του προγραμματιστή μετακινείται από τη συγγραφή του κώδικα της εφαρμογής στην ανάπτυξη ικανών και ευέλικτων μοντέλων.

Στο πρώτο κεφάλαιο παρατίθεται μια επισκόπηση των σημαντικότερων προσπαθειών που έχουν πραγματοποιηθεί μέχρι σήμερα όσον αφορά την ανάπτυξη κατανεμημένων συστημάτων τόσο για την επίλυση πολύπλοκων προβλημάτων όσο και για την παροχή υπηρεσιών.

Στο δεύτερο κεφάλαιο, παρατίθεται μία επισκόπηση των σημαντικότερων αρχιτεκτονικών και εργαλείων λογισμικού για την ανάπτυξη εφαρμογών Ιστού.

Στο τρίτο κεφάλαιο, περιγράφεται η αρχιτεκτονική της πλατφόρμας κινητών πρακτόρων που αναπτύχθηκε με χρήση Υπηρεσιών Ιστού και οι εφαρμογές της. Στο πρώτο μέρος περιγράφονται τα συστατικά της πλατφόρμας των κινητών πρακτόρων και οι οντότητες που την αποτελούν. Στη συνέχεια περιγράφονται οι προσπάθειες που έγιναν για βελτίωση της απόδοσης της πλατφόρμας αλλά και της επέκτασής της ώστε να υποστηρίζει ταυτόχρονα πολλά διαφορετικά κανάλια επικοινωνίας., ενώ γίνεται αποτίμηση της βελτίωσης της απόδοσης της πλατφόρμας. Επίσης, περιγράφονται δύο εφαρμογές που αναπτύχθηκαν πάνω στην πλατφόρμα, και αφορούν ένα σύστημα ηλεκτρονικής ψηφοφορίας και την ανάπτυξη ενός συστήματος κατανεμημένης προσωμοίωσης στοιχειοκεραίας.

Το τέταρτο κεφάλαιο αρχικά αναλύει και περιγράφει την μεταφορά και ανάπτυξη μίας ήδη υπάρχουσας/υλοποιημένης εφαρμογής διπλώματος πρωτεϊνών επάνω από το Πλέγμα (Grid), καθώς και τις αλλαγές και προσθήκες που πραγματοποιήθηκαν για αυτό το εγχείρημα. Πιο αναλυτικά περιγράφεται η αρχιτεκτονική μιας δικτυακής πλεγματικής πύλης η οποία επιτρέπει στους χρήστες την ευκολότερη υποβολή και διαχείριση εργασιών στο Πλέγμα.

Στο πέμπτο κεφάλαιο αναλύεται το ζήτημα της πολυπλοκότητας της ανάπτυξης ή μεταφοράς μιας εφαρμογής σε ένα κατανεμημένο σύστημα. Συγκεκριμένα προτείνεται η υιοθέτηση των αρχών της μοντελοκεντρικής αρχιτεκτονικής προκειμένου να μειωθεί ο χρόνος που απαιτείται για την μεταφορά και/ή ανάπτυξη εφαρμογών στις πλατφόρμες κινητών πρακτόρων, και παρουσιάζονται εργαλεία και βιβλιοθήκες λογισμικού που αναπτύχθηκαν για να διευκολύνουν τη διαδικασία αυτή. Η αποτίμηση της αξίας των εργαλείων αυτών πραγματοποιείται χρησιμοποιώντας μετρικές όπως ο όγκος του κώδικα που χρειάζεται να γράψει ο προγραμματιστής.

Τέλος, στο έκτο κεφάλαιο, παρατίθεται μία επισκόπηση των όσων παρουσιάστηκαν στα κυρίως κεφάλαια της διατριβής. Στόχος της επισκόπησης αυτής είναι η επιβεβαίωση της επίτευξης των στόχων της εργασίας σε σχέση με τις αρχικές απαιτήσεις που είχαν τεθεί.

Επίσης παρουσιάζονται πιθανές και προτεινόμενες μελλοντικές εξελίξεις από τεχνικής πλευράς για την περαιτέρω ανάπτυξη και επέκταση του παρόντος έργου.

2 ΠΛΑΤΦΟΡΜΕΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΕΣ ΚΑΤΑΝΕΜΗΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ

Τα τελευταία χρόνια υπήρξε ραγδαία αύξηση της επίδοσης των υπολογιστών, ενώ παράλληλα το κόστος της μετάδοσης δεδομένων μειώθηκε αισθητά. Με τη ταυτόχρονη άνοδο της ταχύτητας και της ποιότητας των δικτύων υπολογιστών, τόσο τοπικών όσο και μεγαλύτερης κλίμακας, οι συνθήκες αυτές οδήγησαν στη σκέψη για δημιουργία εφαρμογών οι οποίες θα ανταλλάσσουν δεδομένα πάνω από το δίκτυο. Λόγω της πολυπλοκότητας αυτών των εφαρμογών, είναι απαραίτητο να υπάρξει απλότητα και ευελιξία στην αρχιτεκτονική τους. Η ανάγκη αυτή οδήγησε στην ανάπτυξη κατανεμημένων εφαρμογών.

Οι κατανεμημένες εφαρμογές αποτελούνται κατά κανόνα από επιμέρους οντότητες οι οποίες επικοινωνούν μεταξύ τους και συνήθως βρίσκονται σε διαφορετικούς κόμβους, με σκοπό την επίλυση ενός κοινού προβλήματος. Όταν όλες αυτές οι οντότητες μπορούν να θεωρηθούν σαν μια κοινή εφαρμογή, ενώ επικοινωνούν μεταξύ τους χωρίς να γίνεται απευθείας αναφορά σε φυσικές διευθύνσεις (π.χ. διευθύνσεις IP), μπορούμε να μιλάμε για μια κατανεμημένη εφαρμογή. Οι αναφορές σε διευθύνσεις συνήθως υπάρχουν σε κάποιο αρχείο στο δίσκο ή δίνονται στην εφαρμογή σαν είσοδος με κάποιον άλλο τρόπο. Το πρόβλημα που παραμένει είναι πώς η εφαρμογή χρησιμοποιεί τις λειτουργίες χαμηλού επιπέδου για να επικοινωνήσουν οι διεργασίες μεταξύ τους. Η λύση που χρησιμοποιείται είναι η εγκατάσταση λογισμικού σε κάθε κόμβο του δικτύου, το οποίο προσφέρει λειτουργίες υψηλότερου επιπέδου για την επικοινωνία, ενώ χρησιμοποιεί ένα ξεχωριστό σύστημα διευθυνσιοδότησης με το οποίο «κρύβει» τις φυσικές διευθύνσεις πίσω από κάποιες εικονικές. Το λογισμικό αυτό αναφέρεται σαν μεσισμικό, μιας και βρίσκεται ανάμεσα στην εφαρμογή και το δίκτυο, ενώ η πιο σημαντική δυνατότητα του είναι ότι η εφαρμογή δεν επηρεάζεται αν αλλάξει η φυσική διεύθυνση ενός κόμβου, αρκεί το μεσισμικό να μπορεί να χειριστεί τέτοια γεγονότα.

Σε αυτή την παράγραφο θα εξετάσουμε συνοπτικά τις τεχνολογίες μεσισμικού που ξεχώρισαν τα τελευταία χρόνια.

2.1 JINI

Το Jini [1] είναι μια προδιαγραφή που αναπτύχθηκε από τη Sun Microsystems και βασίστηκε στη γλώσσα προγραμματισμού Java. Ορίζει μια αρχιτεκτονική προσανατολισμένη στις υπηρεσίες για δυναμικά περιβάλλοντα. Η πρώτη έκδοση του Jini εμφανίστηκε το 1999, και παρόλο που αρχικά παρουσιάστηκε ως μια τεχνολογία για plug-

and-play συσκευές, χρησιμοποιήθηκε κυρίως για την ανάπτυξη υπηρεσιών. Στόχος του Jini είναι η ελαχιστοποίηση του φόρτου που απαιτείται για τη διαχείριση ενός δικτύου υπηρεσιών.

Η βασική μονάδα του Jini είναι η υπηρεσία. Μία υπηρεσία είναι μια οντότητα που υπάρχει σε ένα δίκτυο και είναι έτοιμη να πραγματοποιήσει κάποια ενέργεια. Η οντότητα αυτή μπορεί να είναι οτιδήποτε, από μια συσκευή μέχρι μια υπηρεσία λογισμικού ή ακόμα και ένα χρήστη. Για παράδειγμα, μια ψηφιακή φωτογραφική μηχανή συμβατή με το Jini μπορεί να προσφέρει μια υπηρεσία για την λήψη φωτογραφιών. Προκειμένου να αναπτυχθεί μια εφαρμογή με το Jini, ένα πρόγραμμα-πελάτης πρέπει να χρησιμοποιήσει αυτές τις υπηρεσίες. Για παράδειγμα ένα πρόγραμμα-πελάτης μπορεί να ζητήσει από την υπηρεσία λήψης φωτογραφίας να του δώσει μία εικόνα, και μετά να την τυπώσει σε έναν εκτυπωτή συμβατό με το Jini.

Η τεχνολογία Jini αποτελείται από δύο κύρια μέρη, το προγραμματιστικό μοντέλο και την υποδομή χρόνου εκτέλεσης.

Η υποδομή χρόνου εκτέλεσης υπάρχει σε δύο διαφορετικές τοποθεσίες: σε υπηρεσίες καταλόγου που υπάρχουν στο δίκτυο και σε συσκευές συμβατές με το Jini. Οι υπηρεσίες καταλόγου βοηθούν την οργάνωση των υπηρεσιών σε ομάδες. Μία ομάδα μπορεί να έχει καταχωρηθεί σε παραπάνω από μία υπηρεσία καταλόγου, επιτρέποντας με αυτόν τον τρόπο να γίνει το σύστημα ανεκτικό σε σφάλματα. Οι υπηρεσίες καταχωρούν τους εαυτούς τους στις υπηρεσίες καταλόγου μέσω μιας διαδικασίας που ονομάζεται ανακάλυψη και συνένωση. Η ανακάλυψη είναι η διαδικασία εύρεσης υπηρεσιών καταλόγου και απόκτησης αναφοράς σε αυτές, ενώ η συνένωση είναι η διαδικασία της δήλωσης μιας υπηρεσίας στην υπηρεσία καταλόγου.

Το προγραμματιστικό μοντέλο του Jini σχεδιάστηκε βασιζόμενο στη διαπίστωση ότι ένα δίκτυο μπορεί να είναι αναξιόπιστο. Το προγραμματιστικό μοντέλο προσφέρει διεπαφές προγραμματισμού για την ανάπτυξη αξιόπιστων συστημάτων. Τα κυριότερα κομμάτια του είναι τα leasing, transaction και κατανεμημένα γεγονότα. Το leasing επιτρέπει διαχείριση του κύκλου ζωής ενός απομακρυσμένου αντικειμένου, και διατηρεί τις αναφορές σε αυτό το αντικείμενο όσο απαιτείται προκειμένου ο garbage collector να μη συλλέξει το αντικείμενο. Η διεπαφή προγραμματισμού των transactions επιτρέπει λειτουργίες όπου μαζικές εντολές και υπηρεσίες είτε να επιτύχουν είτε να αποτύχουν σαν μία μονάδα. Τέλος, το σύστημα κατανεμημένων γεγονότων επιτρέπει σε αντικείμενα να ενημερώνονται μόλις δημιουργείτε κάποιο γεγονός από κάποιο απομακρυσμένο αντικείμενο.

2.2 JXTA

Το JXTA [2] είναι ένα σύνολο από ανοιχτά πρωτόκολλα που επιτρέπουν σε συσκευές συνδεδεμένες σε ένα δίκτυο να επικοινωνούν και να συνεργάζονται με αρχιτεκτονική peer-2-peer. Αυτά τα πρωτόκολλα είναι μικρά και υλοποιούνται και ενσωματώνονται εύκολα σε οποιοδήποτε τύπο συστήματος. Είναι σημαντικό να διευκρινιστεί ότι το JXTA δεν είναι ένα σύνολο διεπαφών προγραμματισμού, αλλά ένα σύνολο προδιαγραφών για τις οποίες οποιοσδήποτε κατασκευαστής υλικού ή λογισμικού μπορεί να προσφέρει τη δικιά του υλοποίηση. Σαν αποτέλεσμα, το JXTA είναι ανεξάρτητο από το υποκείμενο δίκτυο, το οποίο μπορεί να είναι HTTP, TCP/IP, Bluetooth κλπ.

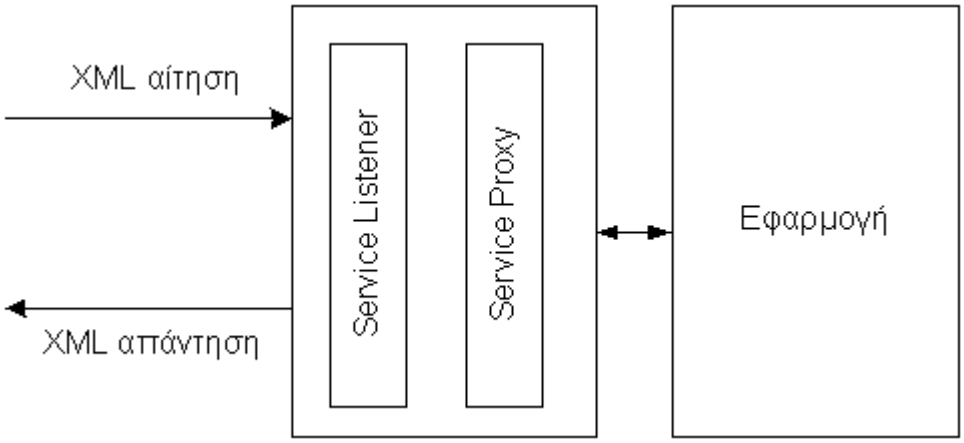
Τα κυριότερα στοιχεία του JXTA περιλαμβάνουν:

Κόμβους (peers). Ένας κόμβος είναι η βασική μονάδα του JXTA. Μπορεί να είναι οποιοσδήποτε τύπος συσκευής συνδεδεμένης στο δίκτυο. Οι κόμβοι χωρίζονται σε δύο κατηγορίες, αυτούς που προσφέρουν τις υπηρεσίες και αυτούς που τις χρησιμοποιούν. Ωστόσο ένας κόμβος μπορεί να ανήκει και στις δύο κατηγορίες.

- Ομάδες κόμβων (peergroups). Μία ομάδα κόμβων είναι ένα αυτοοργανούμενο σύνολο κόμβων. Ένας κόμβος μπορεί να ανήκει σε παραπάνω από μία ομάδα, και μπορεί να χρησιμοποιήσει υπηρεσίες που παρέχονται μόνο μέσα στις ομάδες κόμβων στις οποίες ανήκει.
- Αγωγοί (pipes). Πρόκειται για το εργαλείο επικοινωνίας ανάμεσα σε κόμβους, και είναι παρόμοιο με τα UDP sockets του TCP/IP.
- Διαφημίσεις (advertisements). Κείμενα XML που ανακοινώνουν την ύπαρξη πόρων.
- Ανακάλυψη (discovery). Η διαδικασία της ανακάλυψης πόρων μέσω της διαφήμισης και η αξιοποίηση αυτών των πόρων.

2.3 ΥΠΗΡΕΣΙΕΣ ΙΣΤΟΥ (WEB SERVICES)

Μια υπηρεσία Ιστού [3] είναι ένα σύστημα λογισμικού σχεδιασμένο να υποστηρίζει τη αλληλεπίδραση μηχανών πάνω από ένα δίκτυο. Περιγράφει μια διεπαφή σε μορφή επεξεργάσιμη από μηχανή (χρησιμοποιώντας την τεχνολογία WSDL). Άλλα συστήματα αλληλεπιδρούν με την υπηρεσία Ιστού χρησιμοποιώντας μηνύματα SOAP [5], με έναν τρόπο που ορίζεται από την περιγραφή του. Τα μηνύματα αυτά συνήθως μεταβιβάζονται χρησιμοποιώντας το πρωτόκολλο HTTP και σειριοποίηση XML από κοινού με άλλα πρότυπα σχετικά με τον Ιστό.



Εικόνα 2.1: Γενική αρχιτεκτονική υπηρεσιών ιστού

Μιας και η τεχνολογία των υπηρεσιών ιστού περιγράφει την υπηρεσία, και όχι τον τρόπο με τον οποίο αυτή πρέπει να υλοποιηθεί, είναι δυνατή η υλοποίηση των υπηρεσιών σε διαφορετικά περιβάλλοντα (π.χ. διαφορετική γλώσσα προγραμματισμού και διαφορετικό λειτουργικό σύστημα).

Στις επόμενες παραγράφους του κεφαλαίου θα γίνει μια σύντομη περιγραφή των βασικών τεχνολογιών που χρησιμοποιούν οι υπηρεσίες ιστού.

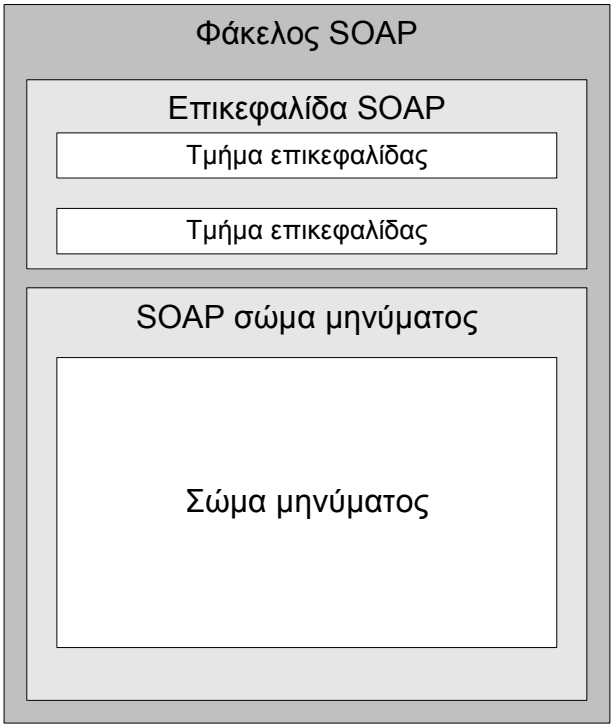
2.3.1 SOAP

Το SOAP (Simple Object Access Protocol – Απλό Πρωτόκολλο Πρόσβασης Αντικειμένου) είναι μια ειδική εφαρμογή της τεχνολογίας XML. Στηρίζεται σε μεγάλο ποσοστό σε πρότυπα XML όπως XML Schema και XML Namespaces για τον καθορισμό και τη λειτουργία του. Η βασική του ιδέα είναι ότι σε οποιαδήποτε κατανεμημένη αρχιτεκτονική κάποια στιγμή είναι απαραίτητη η ανταλλαγή δεδομένων. Το SOAP επιτρέπει αυτή την ανταλλαγή με τη μορφή XML μηνυμάτων τα οποία μεταφέρονται πάνω από το HTTP ή το SMTP.

Υπάρχουν τρία βασικά συστατικά στην προδιαγραφή SOAP: ο φάκελος SOAP, ένα σύνολο κανόνων κωδικοποίησης και ένα μέσο για την αλληλεπίδραση μεταξύ του αιτήματος και της απάντησης. Ο φάκελος SOAP παρέχει τις πληροφορίες για το μήνυμα που κωδικοποιείται σε ένα «φορτίο» SOAP, συμπεριλαμβανομένων των πληροφοριών για τον παραλήπτη και τον αποστολέα, καθώς επίσης και τις λεπτομέρειες για το ίδιο το μήνυμα (π.χ. πώς το μήνυμα πρέπει να υποβληθεί σε επεξεργασία). Η μορφή ενός φακέλου SOAP φαίνεται στην Εικόνα 2.2. Οι κανόνες κωδικοποίησης επιτρέπουν την κωδικοποίηση σύνθετων δομών δεδομένων με τρόπο τέτοιο ώστε να είναι δυνατή η μεταφορά τους με μηνύματα SOAP. Το SOAP υποστηρίζει συγκεκριμένους τύπους δεδομένων (π.χ.

ακέραιους και boolean). Σύνθετοι τύποι, όπως λίστες ή δομές που περιγράφουν π.χ. μια εγγραφή σε μια βάση δεδομένων μπορούν να περιγραφούν. Το μέσο για την αλληλεπίδραση των εφαρμογών είναι, όπως ήδη αναφέρθηκε, συνήθως το πρωτόκολλο HTTP.

Το SOAP έχει δύο εφαρμογές: την απομακρυσμένη κλήση διαδικασιών (RPC) και την ανταλλαγή ηλεκτρονικών εγγράφων (Electronic Document Interchange – EDI). Η πρώτη τεχνολογία είναι παρόμοια με την τεχνολογία XML-RPC, η οποία έχει αναλυθεί ήδη. Δεν είναι τυχαίο ότι αρκετές φορές για την ανάπτυξη υπηρεσιών Ιστού χρησιμοποιείται η τεχνολογία XML-RPC αντί για το SOAP. Η ηλεκτρονική ανταλλαγή εγγράφων είναι η βάση των αυτοματοποιημένων επιχειρησιακών συναλλαγών, και καθορίζει ένα τυποποιημένο σχήμα και μια ερμηνεία των οικονομικών και εμπορικών εγγράφων και μηνυμάτων



Εικόνα 2.2: Δομή φακέλου SOAP

2.3.2 WSDL

Η γλώσσα περιγραφής υπηρεσιών Ιστού [4] (Web Services Description Language - WSDL) παρέχει ένα πρότυπο και ένα σχήμα XML για την περιγραφή των υπηρεσιών Ιστού. Το WSDL επιτρέπει τον χωρισμό της περιγραφής της αφηρημένης λειτουργίας που προσφέρεται από μια υπηρεσία, από τις συγκεκριμένες λεπτομέρειες μιας περιγραφής υπηρεσιών όπως "πώς" και "όπου" εκείνη η λειτουργία προσφέρεται.

Το WSDL περιγράφει διάφορα σημαντικά κομμάτια πληροφοριών που ο πελάτης μιας υπηρεσίας θα μπορούσε να χρειαστεί :

- Το όνομα της υπηρεσίας, συμπεριλαμβανομένου και του URN
- Την τοποθεσία από την οποία είναι προσβάσιμη η υπηρεσία (συνήθως μια διεύθυνση URL)
- Τις μεθόδους που είναι διαθέσιμες για κλήση
- Τον τύπο των παραμέτρων εισόδου και εξόδου για κάθε μέθοδο

Αν και οι παραπάνω πληροφορίες δεν είναι και τόσο χρήσιμες από μόνες τους, ο συνδυασμός τους αναπαριστά το πως ένας πελάτης «βλέπει» την υπηρεσία.

2.3.3 UDDI

Το UDDI (Universal Description, Discovery and Integration - καθολική περιγραφή, ανακάλυψη και ολοκλήρωση) είναι μια προδιαγραφή για κατανεμημένα μητρώα πληροφοριών υπηρεσιών Ιστού. Το UDDI είναι επίσης ένα δημόσια προσιτό σύνολο εφαρμογών της προδιαγραφής που επιτρέπουν στις επιχειρήσεις να καταχωρήσουν τις πληροφορίες για τις υπηρεσίες Ιστού που προσφέρουν έτσι ώστε άλλες επιχειρήσεις να μπορούν να τις βρουν.

Αρχικά η διαχείριση της διαδικασίας της ανακάλυψης των υπηρεσιών ιστού φαίνεται απλή. Αν ένας γνωστός επιχειρησιακός συνεργάτης έχει μια γνωστή πύλη ηλεκτρονικού εμπορίου, δεν υπάρχει ανάγκη να ανακαλυφθεί με κάποιον τρόπο – είναι ήδη γνωστή. Η σιωπηρή υπόθεση, εντούτοις, είναι ότι όλες οι πληροφορίες είναι ήδη γνωστές. Όταν όμως ζητείται να βρεθεί ποιοι επιχειρησιακοί συνεργάτες έχουν ποιες υπηρεσίες, η απάντηση μπορεί εύκολα να γίνει δύσκολη. Μια επιλογή είναι να κληθεί κάθε συνεργάτης στο τηλέφωνο, και να ερωτηθεί, λύση που απορρίπτεται γιατί είναι ιδιαίτερα χρονοβόρα όταν οι επιλογές της αναζήτησης είναι πάρα πολλές.

Ένας άλλος τρόπος να λυθεί αυτό το πρόβλημα είναι μέσω μιας προσέγγισης που χρησιμοποιεί ένα αρχείο περιγραφής υπηρεσιών Ιστού (όπως το WSDL) στην ιστοσελίδα κάθε επιχείρησης. Η λύση αυτή είναι παρόμοια με τη λογική των προγραμμάτων web crawlers που χρησιμοποιούν οι μηχανές αναζήτησης για την ταξινόμηση του κειμένου. Η προσέγγιση αυτή εξαρτάται από τη δυνατότητα για έναν crawler να βρει κάθε ιστοσελίδα και τη θέση του αρχείου περιγραφής υπηρεσιών σε αυτή. Αυτή η κατανεμημένη προσέγγιση είναι ενδεχομένως πιο αποτελεσματική και ευέλικτη, αλλά στερείται ενός μηχανισμού που να εξασφαλίζει τη συνέπεια ως προς τα σχήματα περιγραφής υπηρεσιών και τον εντοπισμό των αλλαγών όταν αυτές πραγματοποιούνται.

Το UDDI υιοθετεί μια μέθοδο που στηρίζεται επάνω σε ένα κατανεμημένο μητρώο των επιχειρήσεων και της περιγραφής των υπηρεσιών που αυτές προσφέρουν χρησιμοποιώντας ένα κοινό σχήμα XML.

Το κύριο συστατικό του προγράμματος UDDI είναι η επιχειρησιακή εγγραφή UDDI, ένα αρχείο XML που χρησιμοποιείται για να περιγράψει μια επιχειρησιακή οντότητα και τις υπηρεσίες Ιστού της. Εννοιολογικά, οι πληροφορίες που παρέχονται σε μια επιχειρησιακή εγγραφή UDDI αποτελούνται από τρία συστατικά:

- "άσπρες σελίδες" συμπεριλαμβανομένης της διεύθυνσης, της επαφής, και των γνωστών προσδιοριστικών
- "κίτρινες σελίδες" συμπεριλαμβανομένων των βιομηχανικών κατηγοριοποιήσεων βασισμένων στις τυποποιημένες ταξινομίες
- "πράσινες σελίδες", οι τεχνικές πληροφορίες για τις υπηρεσίες που εκτίθενται από την επιχείρηση. Οι πράσινες σελίδες περιλαμβάνουν τις αναφορές στις προδιαγραφές για τις υπηρεσίες Ιστού, ενώ υποστηρίζουν και δείκτες σε διάφορα αρχεία και μηχανισμούς ανακάλυψης βασισμένους σε URL.

2.3.4 WEB SERVICES RESOURCE FRAMEWORK

Η εξέλιξη των υπηρεσιών ιστού είναι το Web Services Resource Framework [6](WSRF). Οι υπηρεσίες ιστού δεν πρόσφεραν κάποια μέθοδο ώστε να φυλάσσεται η κατάσταση μιας υπηρεσίας ανάμεσα σε δύο κλήσεις της ίδιας υπηρεσίας. Το WSRF συμπεριλαμβάνει προδιαγραφές που επιτρέπουν να μοντελοποιήσουν την κατάσταση σαν πόρους με κατάσταση (stateful resources). Το WSRF αποτελείται από τις ακόλουθες προδιαγραφές:

- WS-ResourceProperties (WSRF-RP): Προσφέρει μια στάνταρτ μεθοδολογία για την έκφραση των ιδιοτήτων των πόρων.
- WS-ResourceLifetime (WSRF-RL): Επιτρέπει τη διαχείριση του κύκλου ζωής ενός πόρου.
- WS-ServiceGroup (WSRF-SG): Ορίζει πως διαφορετικοί πόροι μπορούν να ομαδοποιηθούν.
- WS-BaseFaults (WSRF-BF): Παρέχει μεθοδολογία για την επεξεργασία σφαλμάτων όταν κάποιο λάθος συμβαίνει κατά την κλήση μίας υπηρεσίας.

Σχετικές με το WSRF είναι και δύο άλλες τεχνολογίες, τα WS-Notification [7] και WS-Addressing [8]. Το WS-Notification ορίζει ένα σύνολο προδιαγραφών που επιτρέπει την αλληλεπίδραση με υπηρεσίες ιστού χρησιμοποιώντας τη μέθοδο της έκδοσης και της

εγγραφής (publish/subscribe), ενώ το WS-Addressing επιτρέπει τον προσδιορισμό του κατάλληλου στιγμιότυπου για την κάθε υπηρεσία ιστού.

2.4 ΚΙΝΗΤΟΙ ΠΡΑΚΤΟΡΕΣ

Η τεχνολογία των πρακτόρων ξεκινάει στις αρχές της δεκαετίας του 1990 με την εμφάνιση τεχνολογιών τις οποίες σήμερα διαχωρίζουμε σε Κινητούς και Ευφυείς Πράκτορες. Η ανάπτυξη Συστημάτων Πολλαπλών Πρακτόρων (Multi Agent Systems – MAS) συνέπεσε με την αύξηση του ενδιαφέροντος για τους κινητούς πράκτορες. Η βασική ιδέα των Συστημάτων Πολλαπλών Πρακτόρων είναι ότι μικρότερες, αυτόνομες οντότητες – οι πράκτορες – μπορούν να αλληλεπιδράσουν μεταξύ τους έτσι ώστε να οδηγήσουν στη λύση ενός πιο σύνθετου και πολύπλοκου προβλήματος. Επομένως, η επικοινωνία ανάμεσα στους πράκτορες και το μοντέλο συνεργασίας τους είναι ένα από τα κυριότερα ζητήματα για αυτόν τον τύπο πρακτόρων.

Η εμφάνιση των Κινητών Πρακτόρων έφερε στο προσκήνιο την τεχνολογία των πρακτόρων. Το 1994 αναπτύχθηκε από την General Magic η γλώσσα TeleScript. Επίκεντρο του ενδιαφέροντος εκείνη την εποχή ήταν οι διερμηνευόμενες γλώσσες (scripting languages), όπως η Tool Command Language (TCL) και η SafeTCL, η οποία προέκυψε από την πρώτη. Οι γλώσσες αυτές επέτρεπαν την πρωτοτυποποίηση (prototyping) και την παραγωγή μεταφέρσιμου κώδικα. Η ενθυλάκωση τέτοιου κώδικα σε μηνύματα ηλεκτρονικού ταχυδρομείου έκανε δυνατή την ανάπτυξη νέων εφαρμογών ηλεκτρονικού ταχυδρομείου, ενώ αναπτύχθηκαν οι βασικές αρχιτεκτονικές και τεχνολογικές αρχές που αποτελούν το βασικό υπόβαθρο της Τεχνολογίας Κινητών Πρακτόρων (Mobile Agent Technology – MAT). Την ίδια χρονική περίοδο έκανε την εμφάνισή της και η γλώσσα προγραμματισμού Java της Sun Microsystems, η οποία πολύ γρήγορα έγινε δημοφιλής στην προγραμματιστική κοινότητα. Σήμερα, τα περισσότερα Συστήματα Κινητών Πρακτόρων είναι υλοποιημένα σε Java.

Ένα από τα προτερήματα της TeleScript ήταν ότι ενσωμάτωνε ένα πλήρες Περιβάλλον Εκτέλεσης Κινητών Πρακτόρων (Mobile Agent Execution Environment). Το περιβάλλον αυτό είχε σχεδιασθεί έτσι ώστε να επιτρέπει την υλοποίηση μεθόδων απομακρυσμένου προγραμματισμού (Remote Programming). Η βασική ιδέα ήταν ότι μικρά κομμάτια κώδικα θα μεταφερόντουσαν εκεί που βρισκόντουσαν μεγάλες ποσότητες δεδομένων, μειώνοντας έτσι τον φόρτο του δικτύου. Η TeleScript επέτρεπε την ύπαρξη αυτόνομων πρακτόρων, οι οποίοι μπορούσαν να επικοινωνήσουν μεταξύ τους ή με τον χρήστη και να μεταναστεύσουν σε απομακρυσμένους κόμβους προκειμένου να εκτελέσουν την εργασία που έχουν

αναλάβει. Ο κάθε πράκτορας είχε τη δυνατότητα να αναστείλει τη λειτουργία του, να μεταφέρει τον κώδικα, την κατάσταση εκτέλεσης και τα δεδομένα του και να μεταφερθεί σε κάποιον άλλο κόμβο για να συνεχίσει τη λειτουργία του.

Ο γενικότερος ενθουσιασμός και η αναπτυξιακή προσπάθεια που υπάρχουν σήμερα στην περιοχή της τεχνολογίας των κινητών πρακτόρων οφείλεται κατά μεγάλο βαθμό στη γλώσσα προγραμματισμού Java. Η Java υποστηρίζει την τεχνολογία κινητού κώδικα μέσω των δυνατοτήτων της για σειριοποίηση αντικειμένων (object serialization), η οποία επιτρέπει το δυναμικό φόρτωμα κώδικα κατά τη διάρκεια εκτέλεσης ενός προγράμματος.

2.4.1 ΚΥΡΙΑ ΣΥΣΤΑΤΙΚΑ ΠΛΑΤΦΟΡΜΑΣ ΚΙΝΗΤΩΝ ΠΡΑΚΤΟΡΩΝ

Η υποδομή μιας πλατφόρμας κινητών πρακτόρων παρέχει την υποστήριξη υπηρεσιών κινητικότητας και επικοινωνίας κατά τον χρόνο εκτέλεσης. Θα πρέπει να είναι δυνατή η μετανάστευση ενός πράκτορα από ένα διακομιστή σε ένα άλλο ώστε να μπορεί να έχει πρόσβαση σε δεδομένα και πληροφορίες τις οποίες να μεταφέρει πίσω στην αρχική του θέση. Μια πλατφόρμα κινητών πράκτορα μπορεί να υλοποιηθεί σε πολλές γλώσσες προγραμματισμού αλλά, όπως ήδη αναφέρθηκε, τα τελευταία χρόνια η Java έχει αποδειχτεί ως η πλέον κατάλληλη γι' αυτό το σκοπό εξαιτίας της πολυνηματικής επεξεργασίας, της σειριοποίησης αντικειμένων και των δικτυακών διεπαφών που προσφέρει.

Οι κινητοί πράκτορες είναι οντότητες λογισμικού, οι οποίες μπορούν αυτόνομα να μεταναστεύσουν από έναν εξυπηρετητή σε κάποιον άλλο κατά τη διάρκεια της εκτέλεσής τους. Οι πράκτορες αποτελούνται από τον κώδικα, τα δεδομένα καθώς επίσης και από την κατάσταση εκτέλεσής τους. Η μεταφορά της κατάστασης εκτέλεσης επιτρέπει σε έναν πράκτορα να συνεχίσει την εργασία του αμέσως μόλις μεταναστεύσει προς έναν άλλο εξυπηρετητή, από το σημείο όπου είχε σταματήσει πριν τη μετανάστευση. Στην περίπτωση της Java, η εικονική μηχανή της δεν είναι ικανή να ανακτήσει το σωρό εκτέλεσης ενός νήματος. Για αυτόν τον λόγο, σε περίπτωση μετανάστευσης ενός πράκτορα μόνο οι ψηφιολέξεις (bytecodes) της κλάσης και τα δεδομένα του στιγμιότυπου της κλάσης μπορούν να μεταφερθούν στο απομακρυσμένο εξυπηρετητή δίχως το σωρό εκτέλεσης. Μια λύση για αυτό το πρόβλημα είναι οι κινητοί πράκτορες να αποθηκεύουν τις απαραίτητες πληροφορίες κατάστασης σε μεταβλητές μελών της κλάσης, που επιτρέπουν σε μια μέθοδο που καθορίζεται ως σημείο εισόδου να εξετάσει αυτές τις μεταβλητές και να προχωρήσει ανάλογα με την κατάσταση στην οποία βρίσκονται οι υπολογισμοί [9]. Αυτό το σχήμα αναφέρεται ως "ασθενής μετανάστευση" σε αντίθεση με τη σχήμα της "ισχυρής μετανάστευσης" όπου η μεταφορά του σωρού εκτέλεσης επιτρέπει στην υποδομή να

επαναστιγμιοτυπήσει τον πράκτορα από το σημείο που είχε σταματήσει την εκτέλεση προτού κληθεί η μέθοδος "move" στον προηγούμενο εξυπηρετητή.

Το τρίτο μέρος ενός κινητού πράκτορα αποτελείται από τις ιδιότητες, οι οποίες περιγράφουν τις απαιτήσεις και την ιστορία του στην υποδομή. Αυτό περιλαμβάνει στοιχεία όπως ένα μοναδικό προσδιοριστή, τον ιδιοκτήτη του πράκτορα ως μια διεύθυνση για τα ενδιάμεσα αποτελέσματα, μηνύματα λάθους για τη σωστή συμπεριφορά ενός πράκτορα, τον τόπο και χρόνο προέλευσης και το ιστορικό των μετακινήσεών του. Αυτές οι πληροφορίες μπορούν να προσαρμοστούν σύμφωνα με την προέλευση ή το σκοπό του εκάστοτε πράκτορα. Το πλεονέκτημα αυτού του σχήματος είναι ότι κανένας εξυπηρετητής δεν χρειάζεται τις πλήρεις πληροφορίες για όλους τους πιθανούς προορισμούς για τους κινητούς πράκτορες.

Τέλος, αναγκαία είναι η ύπαρξη μιας υπηρεσίας καταλόγου για την εύρεση της θέσης ενός αντικειμένου. Οποιαδήποτε λογική υπηρεσία καταλόγου για τους κινητούς πράκτορες πρέπει να χρησιμοποιεί αποδοτικούς μηχανισμούς για την παρακολούθηση των πρακτόρων ώστε να ανακτάται η τρέχουσα θέση τους.

2.5 GRID

Η τεχνολογία Υπολογιστικού Πλέγματος (Grid Computing) είναι μια καινούργια τεχνολογία που έρχεται να προσφέρει μια νέα αρχιτεκτονική για την επίλυση προβλημάτων μεγαλύτερης κλίμακας, αλλά και να λύσει προβλήματα συνεργασίας ανάμεσα σε ανομοιογενείς πόρους. Η κεντρική ιδέα του Πλέγματος είναι η δημιουργία ενός δικτύου το οποίο θα επιτρέπει το διαμερισμό των πόρων στους χρήστες του δικτύου. Η τεχνολογία αυτή υπόσχεται πολλά, και πολλές αρχιτεκτονικές έχουν προταθεί για την υλοποίησή της, με κάποιες να είναι πιο διαδεδομένες από άλλες, αλλά χωρίς κάποια να είναι ο απόλυτος κυρίαρχος στο χώρο, με αποτέλεσμα το Πλέγμα να είναι ένα νέο, συναρπαστικό, πεδίο έρευνας.

Μια κατηγοριοποίηση που υπάρχει ανάμεσα στα είδη πλεγμάτων είναι τα Υπολογιστικά πλέγματα (Computational Grids), τα Πλέγματα Δεδομένων (Data Grids) και τα Πλέγματα Υπηρεσιών (Service Grids) [46]. Εκτός από την κατανομή των κατηγοριών πλέγματος βάση των λειτουργιών τους, υπάρχει και κατηγοριοποίησή τους βάσει του μοντέλου της αρχιτεκτονικής τους, αν έχουν ιεραρχική δομή ή βασίζονται σε μοντέλο διασύνδεσης peer-to-peer. Άλλη κατηγορία είναι αυτή των δυναμικά κατανεμημένων υποδομών (Scavenging Grids), όπου το πλέγμα «κλέβει» πόρους από τον υπολογιστή όταν αυτός είναι ανενεργός.

Σε αυτή την παράγραφο περιγράφουμε τις κατηγορίες Πλέγματος και την εξέλιξη των συστημάτων πλέγματος, χωρίζοντάς τη σε τρεις γενεές: συστήματα πρώτης γενεάς που ήταν οι πρόδρομοι του πλέγματος έτσι όπως το αναγνωρίζουμε σήμερα, συστήματα δεύτερης γενεάς εστιάζοντας στο μεσισμικό για υποστήριξη τα υπολογισμών και δεδομένων μεγάλης κλίμακας, και συστήματα τρίτης γενεάς όπου ιδιαίτερη έμφαση δίνεται στη παγκόσμια συνεργασία (global collaboration), μια προσέγγιση προσανατολισμένη στις υπηρεσίες και σε ζητήματα του στρώματος πληροφοριών. Ειδικότερα, συζητάμε τη σχέση μεταξύ του πλέγματος και του Παγκόσμιου Ιστού (World Wide Web), και πως οι τεχνολογίες Ιστού παρέχουν τη βάση για την επόμενη γενεά του πλέγματος.

2.5.1 ΠΡΩΤΗ ΓΕΝΕΑ

Οι πρόωρες προσπάθειες για τη δημιουργία πλέγματος άρχισαν ως προγράμματα για να συνδέσουν περιοχές με δυνατότητες supercomputing. Αυτή η προσέγγιση ήταν γνωστή όπως metacomputing. Η προέλευση του όρου θεωρείται το πρόγραμμα CASA, ένα από τα Gigabit testbed που υπήρχαν γύρω στο 1989. Ο Larry Smarr, πρώην διευθυντής της NCSA, είναι αυτός που αναγνωρίζεται ως αυτός που διέδωσε τον όρο “Grid” έκτοτε [22].

Από την αρχή μέχρι τα μέσα της δεκαετίας του '90 κάνουν την εμφάνισή τους τα πρώτα περιβάλλοντα πλέγματος. Ο στόχος αυτών των πρώτων προγραμμάτων metacomputing ήταν να παρασχεθούν οι υπολογιστικοί πόροι σε μια σειρά εφαρμογών υψηλής απόδοσης. Δύο πρωτοπόρα προγράμματα στη τεχνολογία αυτή ήταν το FAFNER [23] και το I-WAY [24]. Αυτά τα προγράμματα διαφέρουν από πολλές απόψεις, αλλά και τα δύο έπρεπε να υπερνικήσουν διάφορα παρόμοια εμπόδια, συμπεριλαμβανομένων των επικοινωνιών, της διαχείρισης των πόρων, και του χειρισμού των απομακρυσμένων δεδομένων, να είναι σε θέση να εργαστούν αποδοτικά και αποτελεσματικά. Τα δύο προγράμματα προσπάθησαν επίσης να παρέχουν τους πόρους των συστημάτων metacomputing από αντίθετες άκρες των υπολογιστικών δυνατοτήτων. Το FAFNER ήταν σε θέση να εκτελεσθεί σε οποιοδήποτε τερματικό σταθμό με περισσότερα από 4 MB μνήμης, ενώ το I-WAY ήταν ένας τρόπος για την ενοποιημένη χρήση των πόρων μεγάλων κέντρων supercomputing.

Τόσο το FAFNER όσο και το I-WAY ήταν ιδιαίτερα καινοτόμα και επιτυχή. Κάθε πρόγραμμα ήταν πρωτοποριακό και βοήθησε στην προετοιμασία του εδάφους για πολλά από τα προγράμματα πλέγματος δεύτερης γενεάς. Το FAFNER ήταν ο πρόδρομος προγραμμάτων όπως το SETI@home [28] και του distributed.net [27], ενώ το I-WAY άνοιξε το δρόμο για το Globus [25] και το Legion [26].

2.5.2 ΔΕΥΤΕΡΗ ΓΕΝΕΑ

Στις πρόωρες προσπάθειες για τη δημιουργία υπολογιστικών πλεγμάτων δόθηκε ιδιαίτερη έμφαση στην ανάγκη να συνδεθούν διάφορα κέντρα supercomputing. Το πρόγραμμα I-WAY επιτυχώς πέτυχε αυτόν τον στόχο. Σήμερα η υποδομή πλέγματος είναι σε θέση να συνδέσει μαζί κάτι παραπάνω από μερικά ειδικευμένα κέντρα supercomputing. Υπάρχουν διάφορα προγράμματα και τεχνολογίες που έχουν βοηθήσει στη διάδοση του πλέγματος, συμπεριλαμβανομένης της λήψης των τεχνολογιών των δικτύων εύρους ζώνης και της θέσπισης προτύπων. Όλες αυτές οι τεχνολογίες επιτρέπουν στο πλέγμα να αντιμετωπισθεί ως μία βιώσιμη κατανεμημένη υποδομή σε παγκόσμια κλίμακα η οποία μπορεί να υποστηρίξει διαφορετικές εφαρμογές που απαιτούν υπολογισμούς και δεδομένα μεγάλης κλίμακας. Αυτό το όραμα του πλέγματος παρουσιάστηκε στο [29] και θεωρούμε αυτό ως δεύτερη γενεά, άποψη που απεικονίζεται σε πολλές από τις σημερινές εφαρμογές πλέγματος.

Υπάρχουν τρία κύρια ζητήματα που έπρεπε αντιμετωπισθούν:

- Ετερογένεια. Το Πλέγμα αναμειγνύει πολλαπλούς πόρους ετερογενούς φύσης. Οι πόροι αυτοί μπορεί να βρίσκονται σε πολλές διαφορετικές διαχειριστικές περιοχές, και σε παγκόσμια κλίμακα.
- Εξελιξιμότητα. Ένα Πλέγμα μπορεί να μεγαλώσει από λίγους πόρους σε εκατομμύρια. Αυτό έχει σαν αποτέλεσμα την εμφάνιση προβλημάτων πτώσης της απόδοσης όπως μεγαλώνει το μέγεθος του Πλέγματος. Σαν συνέπεια, οι εφαρμογές που απαιτούν ένα μεγάλο πλήθος πόρων πρέπει να σχεδιάζονται έτσι ώστε να είναι ανεκτικές σε πτώση της απόδοσης και να εξερευνούν την τοπικότητα των προσβάσιμων πόρων. Επιπλέον, η αύξηση της κλίμακας εμπλέκει τη συνεργασία διαφορετικών οργανισμών, με αποτέλεσμα την ύπαρξη ετερογένειας αλλά και την εμφάνιση της ανάγκης για τη δημιουργία ενός συστήματος πιστοποίησης και αντιμετώπισης προβλημάτων εμπιστοσύνης ανάμεσα στους φορείς. Τέλος, εφαρμογές μεγάλης κλίμακας μπορεί να προκύψουν και από τη σύνδεση επιμέρους εφαρμογών, με αποτέλεσμα την αύξηση της πολυπλοκότητας του «διαλόγου» ανάμεσα στα συστήματα.
- Προσαρμοστικότητα. Σε ένα πλέγμα, η βλάβη ενός πόρου είναι ο κανόνας, και όχι η εξαίρεση. Στη πραγματικότητα, υπάρχουν τόσοι πολλοί πόροι σε ένα πλέγμα, έτσι ώστε η πιθανότητα βλάβης κάποιου πόρου είναι από τη φύση της υψηλή. Οι διαχειριστές των πόρων ή οι εφαρμογές πρέπει να έχουν δυναμική συμπεριφορά έτσι

ώστε να επιτυγχάνουν την καλύτερη δυνατή επίδοση από τις διαθέσιμες υπηρεσίες και πόρους.

Το μεσισμικό θεωρείται γενικά το στρώμα του λογισμικού που τοποθετείται μεταξύ του λειτουργικού συστήματος και των εφαρμογών, παρέχοντας ποικίλες υπηρεσίες που απαιτούνται από μια εφαρμογή για να λειτουργήσει σωστά. Πρόσφατα, το μεσισμικό έχει επανεμφανιστεί ως μέσο ενσωμάτωσης των εφαρμογών λογισμικού που τρέχουν σε κατανεμημένα ετερογενή περιβάλλοντα. Σε ένα Πλέγμα, το μεσισμικό χρησιμοποιείται για να κρύψει την ετερογενή φύση και να παρέχει στους χρήστες και τις εφαρμογές ένα ομοιογενές περιβάλλον με την παροχή ενός συνόλου τυποποιημένων διεπαφών σε ποικίλες υπηρεσίες.

Ο καθορισμός και η χρησιμοποίηση των προτύπων είναι βασικοί παράγοντες στην αντιμετώπιση της ετερογένειας. Τα συστήματα χρησιμοποιούν ποικίλα πρότυπα και APIs, με συνέπεια την ανάγκη τη μετατροπή υπηρεσιών και τις εφαρμογών στο μεγάλο αριθμό ηλεκτρονικών υπολογιστών που χρησιμοποιούνται σε ένα περιβάλλον πλέγματος. Σαν γενική αρχή, τα συμφωνηθέντα σχήματα ανταλλαγής βοηθούν να μειώσουν την πολυπλοκότητα, επειδή απαιτούνται N μετατροπείς για να επιτρέψουν σε N συστατικά να επικοινωνήσουν μέσω ενός προτύπου, σε αντιδιαστολή με τους N^2 μετατροπείς για να επικοινωνήσουν το ένα με το άλλο.

Κατά τη διάρκεια ανάπτυξης των συστημάτων Πλέγματος δεύτερης γενεάς έγιναν εμφανείς οι απαιτήσεις που υπάρχουν για την υποδομή του:

- Ιεραρχία διαχείρισης (Administrative Hierarchy). Η ιεραρχία της διαχείρισης είναι ο τρόπος που κάθε περιβάλλον πλέγματος διαιρεί τον εαυτό του έτσι ώστε να αντιμετωπίσει τη παγκόσμια φύση του. Η ιεραρχία της διαχείρισης, παραδείγματος χάριν, καθορίζει πώς θα ρέουν πληροφορίες διαχειριστικού χαρακτήρα μέσω του πλέγματος.
- Υπηρεσίες επικοινωνίας (Communication Services). Οι ανάγκες των εφαρμογών που χρησιμοποιούν το περιβάλλον του πλέγματος ποικίλλουν από αξιόπιστη επικοινωνία σημείου με σημείο, μέχρι αναξιόπιστη αναμετάδοση (multicast). Η υποδομή της επικοινωνίας χρειάζεται να υποστηρίζει πρωτόκολλα που χρησιμοποιούνται για μεταφορά μεγάλου όγκου δεδομένων, ροές δεδομένων (data streams), επικοινωνία ομάδων, και αυτές που χρησιμοποιούνται από τεχνολογίες κατανεμημένων αντικειμένων. Οι υπηρεσίες του δικτύου πρέπει να παρέχουν στο πλέγμα σημαντικές παραμέτρους ποιότητας της υπηρεσίας, όπως το εύρος ζώνης, η ανεκτικότητα στο σφάλμα κλπ.

- Υπηρεσίες πληροφοριών (Information Services). Ένα Πλέγμα είναι ένα δυναμικό περιβάλλον όπου η θέση και ο τύπος των διαθέσιμων υπηρεσιών αλλάζουν συνεχώς. Ένας σημαντικός στόχος είναι όλοι οι πόροι να είναι προσιτοί σε οποιαδήποτε διεργασία μέσα στο σύστημα, αδιαφορώντας για τη σχετική θέση του χρήστη των πόρων. Είναι απαραίτητο να παρασχεθούν οι μηχανισμοί για να επιτρέψει ένα περιβάλλον στο οποίο οι πληροφορίες για το πλέγμα λαμβάνονται αξιόπιστα και εύκολα από εκείνες τις υπηρεσίες που ζητούν τις πληροφορίες. Οι υπηρεσίες πληροφοριών Πλέγματος (εγγραφή και κατάλογος) παρέχουν τους μηχανισμούς για την καταγραφή και τη λήψη των πληροφοριών σχετικά με τη δομή, τους πόρους, τις υπηρεσίες, τη θέση και τη φύση του περιβάλλοντος.
- Υπηρεσίες ονομάτων (Naming Services). Σε ένα πλέγμα, όπως σε οποιοδήποτε άλλο κατανεμημένο σύστημα, τα ονόματα χρησιμοποιούνται για την αναφορά σε μια ευρεία ποικιλία αντικειμένων όπως υπολογιστές, υπηρεσίες ή δεδομένα. Η υπηρεσία ονομάτων παρέχει ένα ομοιόμορφο χώρο ονομάτων στο κατανεμημένο περιβάλλον. Χαρακτηριστικές υπηρεσίες ονομάτων παρέχονται από το διεθνές σχήμα ονομασίας X.500 ή το DNS.
- Κατανεμημένο σύστημα αρχείων (Distributed File System). Οι κατανεμημένες εφαρμογές, τις περισσότερες φορές, απαιτούν την πρόσβαση στα αρχεία που κατανέμονται μεταξύ πολλών εξυπηρετητών. Ένα κατανεμημένο σύστημα αρχείων είναι επομένως ένα βασικό συστατικό σε ένα κατανεμημένο σύστημα. Από την άποψη των εφαρμογών είναι σημαντικό ένα κατανεμημένο σύστημα αρχείων να μπορεί να παρέχει ένα ομοιόμορφο και κοινό χώρο ονομάτων, να υποστηρίζει μια σειρά από πρωτοκόλλα εισόδου/εξόδου (I/O) αρχείων, να απαιτεί ελάχιστη ή καμία τροποποίηση των προγραμμάτων, και να παρέχει μέσα που να επιτρέπουν τη βελτιστοποίηση της απόδοσης (όπως η χρήση ενδιάμεσης μνήμης).
- Ασφάλεια και Έγκριση (Security and Authorisation). Οποιοδήποτε κατανεμημένο σύστημα περιλαμβάνει και τις τέσσερις πτυχές της ασφάλειας: εμπιστευτικότητα (confidentiality), ακεραιότητα (integrity), επικύρωση (authentication) και υπευθυνότητα (accountability). Η ασφάλεια μέσα σε ένα περιβάλλον Πλέγματος είναι ένα σύνθετο ζήτημα που απαιτεί από διαφορετικούς πόρους που αντιμετωπίζονται αυτόνομα να αλληλεπιδράσουν με έναν τρόπο που δεν προσκρούει στη δυνατότητα χρησιμοποίησης τους και που δεν εισάγει τις κενά/σφάλματα ασφάλειας στα μεμονωμένα συστήματα ή τα περιβάλλοντα

συνολικά. Μια υποδομή ασφάλειας είναι το κλειδί στην επιτυχία ή την αποτυχία ενός περιβάλλοντος Πλέγματος.

- Κατάσταση συστήματος και ανοχή σφαλμάτων (System Status and Fault Tolerance). Για να παρέχουν ένα αξιόπιστο και ισχυρό περιβάλλον είναι σημαντικό να παρέχονται τα μέσα για την επίβλεψη των πόρων και των εφαρμογών.
- Διαχείριση πόρων και προγραμματισμός (Resource Management and Scheduling). Η διαχείριση του χρόνου του επεξεργαστή, της μνήμης, του δικτύου, της αποθήκευσης, και άλλων συστατικών σε ένα Πλέγμα είναι σαφώς σημαντικές. Ο γενικός στόχος είναι ο αποδοτικός και αποτελεσματικός προγραμματισμός των εφαρμογών που πρέπει να χρησιμοποιήσουν τους διαθέσιμους πόρους στο κατανεμημένο περιβάλλον. Από την πλευρά του χρήστη, η διαχείριση των πόρων και ο προγραμματισμός πρέπει να είναι διαφανείς και η αλληλεπίδρασή της διαχείρισης με τον χρήστη πρέπει να περιοριστεί στην υποβολή της εφαρμογής. Είναι σημαντικό σε ένα Πλέγμα ότι οι υπηρεσίες διαχείρισης των πόρων και προγραμματισμού να μπορούν να αλληλεπιδράσουν με εκείνες που έχουν εγκατασταθεί τοπικά.
- Γραφική διεπαφή χρήστη (GUI). Οι διεπαφές που επιτρέπουν την πρόσβαση στις υπηρεσίες και τους πόρους πρέπει να είναι εύχρηστοι και εύκολα κατανοητοί από τους χρήστες, καθώς και ετερογενείς. Συνήθως τόσο η διεπαφή του χρήστη όσο και του διαχειριστή βασίζονται σε διεπαφές Ιστού.

Υπάρχει ένα συνεχώς αυξανόμενο πλήθος προγραμμάτων που σχετίζονται με το Πλέγμα. Τα προγράμματα αυτά αντιμετωπίζουν περιοχές όπως η υποδομή, βασικές υπηρεσίες, συνεργατικότητα, ειδικές περιοχές, καθώς και δικτυακές πύλες που επιτρέπουν την πρόσβαση σε συγκεκριμένες περιοχές. Τα πιο σημαντικά από αυτά είναι το Globus και το Legion. Και τα δύο αντιμετωπίζουν τα ζητήματα που αναφέρθηκαν παραπάνω, το καθένα με τη δική του προσέγγιση. Το Globus 2 είναι η εξέλιξη του Globus1 που χρησιμοποιήθηκε στο I-WAY, ενώ η κυκλοφορία του Globus 3 οριοθετεί την τρίτη γενεά Πλέγματος. Από την άλλη, το Legion χρησιμοποιεί μια πιο αντικειμενοστραφή όψη του προβλήματος.

Πέρα από τα δύο παραπάνω προγράμματα, υπάρχουν πολλές άλλες τεχνολογίες και προγράμματα που αντιμετωπίζουν τα επιμέρους ζητήματα, και που μπορούν να συνδυαστούν για την ανάπτυξη ενός πλέγματος. Για παράδειγμα, η CORBA επιτρέπει τη χρήση κατανεμημένων αντικειμένων, ενώ τεχνολογίες όπως το Jini και το RMI την εκτέλεση ομάδων διεργασιών σε κατανεμημένα συστήματα. Πακέτα όπως το Condor [32] ή η Sun Grid Engine [33] επιτρέπουν τον προγραμματισμό των διεργασιών, ενώ το

Nimrod/G προτείνει μια λύση για ένα κατανεμημένο σύστημα αρχείων. Την διεπαφή με το χρήστη αναλαμβάνουν τα Grid Portals όπως το NPACI HotPage [34]. Άλλα προγράμματα με ενδιαφέρον είναι το Cactus [35], το UNICORE [37], το DataGrid [36] και το WebFlow [38], [39]. Τέλος, αξίζει να σημειωθεί ότι για την αντιμετώπιση των προβλημάτων συντονισμού και συνεργασίας μεγάλου πλήθους κόμβων έχει προταθεί η χρήση αρχιτεκτονικών peer-to-peer.

Συμπερασματικά, στη δεύτερη γενεά ο πυρήνας του λογισμικού για το Πλέγμα έχει εξελιχθεί από τις πρόωρες μορφές του, όπως το Globus (GT1) και το Legion -οι οποίες αφιερώθηκαν στην παροχή υπηρεσιών στις μεγάλες και υπολογιστικά απαιτητικές εφαρμογές υψηλής απόδοσης - κατευθείαν στη γενικότερη και ανοικτή επέκταση Globus (GT2) και Avaki. Παράλληλα με αυτό τον πυρήνα του λογισμικού, η δεύτερη γενεά είδε επίσης την ανάπτυξη μιας σειράς από συνοδευτικά εργαλεία, που αναπτύχθηκαν για να παρέχουν υπηρεσίες υψηλότερου επιπέδου τόσο στους χρήστες όσο και στις εφαρμογές, και εκτείνεται από προγραμματιστές του χρόνου (schedulers) των πόρων και τους μεσίτες αντικειμένων μέχρι διεπαφές χρήστη και δικτυακές πύλες. Οι P2P τεχνικές έχουν προκύψει επίσης κατά τη διάρκεια αυτής της περιόδου.

2.5.3 ΤΡΙΤΗ ΓΕΝΕΑ

Η δεύτερη γενεά παρείχε τη διαλειτουργικότητα που ήταν απαραίτητη για την επίτευξη υπολογισμών μεγάλης κλίμακας. Δεδομένου ότι οι περαιτέρω λύσεις Πλέγματος εξερευνήθηκαν, άλλες πτυχές της εφαρμοσμένης μηχανικής του Πλέγματος έγιναν προφανείς. Προκειμένου να δομηθούν νέες εφαρμογές πλέγματος ήταν επιθυμητό τα υπάρχοντα συστατικά και οι πηγές πληροφοριών να είναι επαναχρησιμοποιούμενα και να συγκεντρώνονται κατά τρόπο εύκαμπτο. Η λύση που υιοθετείται ολοένα και περισσότερο είναι προσανατολισμένη προς το μοντέλο των υπηρεσιών και της αυξανόμενης προσοχής στα μεταδεδομένα (metadata) – αυτά είναι τα δύο βασικά χαρακτηριστικά των συστημάτων τρίτης γενεάς. Στην πραγματικότητα η ίδια η - προσανατολισμένη προς τις υπηρεσίες - προσέγγιση έχει τις επιπτώσεις στις πληροφορίες: η εύκαμπτη συγκέντρωση των πόρων του Πλέγματος σε μια εφαρμογή Πλέγματος απαιτεί πληροφορίες για τη λειτουργία, τη διαθεσιμότητα και τις διεπαφές των διάφορων συστατικών, και αυτές οι πληροφορίες πρέπει να έχουν μια συμφωνηθείσα ερμηνεία που να μπορεί να υποβληθεί σε επεξεργασία από κάποιο υπολογιστή.

Εκτιμώντας ότι το πλέγμα είχε περιγραφεί παραδοσιακά από την άποψη των μεγάλης κλίμακας δεδομένων και υπολογισμών, η μετατόπιση της προσοχής στην τρίτη γενεά ήταν

προφανής από τις νέες περιγραφές. Ειδικότερα, οι όροι της «κατανεμημένης συνεργασία» (distributed collaboration) και του εικονικού οργανισμού (virtual organization) υιοθετήθηκαν στο έγγραφο της ανατομίας του Πλέγματος [40]. Η τρίτη γενεά είναι μια πιο ολιστική άποψη του υπολογιστικού Πλέγματος και μπορεί να ειπωθεί ότι εξετάζει την υποδομή για την e-επιστήμη (e-Science) – ένας όρος που υπενθυμίζει σε μας τις απαιτήσεις (να κάνει τη νέα επιστήμη), και του e-επιστήμονα (e-Scientist) παρά την τεχνολογία.

2.6 ΕΦΑΡΜΟΓΕΣ ΙΣΤΟΥ

Ο όρος «Εφαρμογές Ιστού» (Web applications) αναφέρεται στις εφαρμογές οι οποίες μπορούν να χρησιμοποιηθούν μέσα από ένα φυλλομετρητή ιστού (Web browser), μέσω ενός δικτύου υπολογιστών όπως το Internet ή ένα Intranet. Ενώ οι απλές ιστοσελίδες επιστρέφουν στατικό περιεχόμενο, οι εφαρμογές ιστού επιστρέφουν δυναμικές απαντήσεις, οι οποίες εξαρτώνται από τις ενέργειες του χρήστη. Προκειμένου να διευκολυνθεί η γρήγορη ανάπτυξη εφαρμογών ιστού έχουν αναπτυχθεί διάφορες πλατφόρμες οι οποίες προσφέρουν διεπαφές προγραμματισμού (Application Programming Interfaces – APIs).

2.6.1 ΠΛΑΤΦΟΡΜΕΣ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΙΣΤΟΥ

Οι πλατφόρμες ανάπτυξης εφαρμογών ιστού [47] είναι εργαλεία που επιτρέπουν στον προγραμματιστή να αναπτύξει γρήγορα και εύκολα εφαρμογές ιστού. Υπάρχουν πολλές διαφορετικές προσεγγίσεις, οι οποίες μπορούν να κατηγοριοποιηθούν βάσει της γλώσσας προγραμματισμού, το σχεδιαστικό πρότυπο (design pattern) κλπ. Για παράδειγμα, για μια απλή εφαρμογή που παρουσιάζει μια λίστα από τις συναντήσεις ενός χρήστη, ο προγραμματιστής συνήθως ορίζει τα αντικείμενα που θα χρησιμοποιήσει. Έτσι μπορεί να ορίσει το αντικείμενο «Συνάντηση» το οποίο περιέχει τίτλο, περιγραφή, ημερομηνία και ώρα. Η πλατφόρμα ανάπτυξης μέσω των εργαλείων που προσφέρει, θα δημιουργήσει αυτόματα τη λειτουργικότητα που θα επιτρέψει στον χρήστη να προσθέσει, επεξεργαστεί, διαγράψει και να δει σε μορφή λίστας τις εγγραφές του αντικειμένου «Συνάντηση».
Σε αυτή την παράγραφο θα παρουσιάσουμε τις πιο διαδεδομένες πλατφόρμες ανάπτυξης εφαρμογών ιστού.

2.6.2 APACHE STRUTS

Το Struts [48] είναι μια πλατφόρμα ανάπτυξης εφαρμογών ιστού μου βασίζεται στη γλώσσα προγραμματισμού Java και χρησιμοποιεί αρχιτεκτονική model-view-controller (MVC). Η αρχιτεκτονική αυτή χωρίζει την εφαρμογή σε τρία μέρη: το μοντέλο (model), την όψη

(view) και τον ελεγκτή (controller). Ο ελεγκτής είναι το κομμάτι της εφαρμογής που είναι υπεύθυνο για να χειρίζεται την είσοδο των δεδομένων. Η επεξεργασία των δεδομένων γίνεται μέσα στο κομμάτι του μοντέλου, ενώ την αναπαράσταση των αποτελεσμάτων αναλαμβάνει το συστατικό της όψης.

Όπως ήδη αναφέρθηκε, το Struts βασίζεται στη γλώσσα προγραμματισμού Java. Πιο συγκεκριμένα, τα κομμάτια του μοντέλου και του ελεγκτή είναι ένα σύνολο από servlets και Java Bean, ενώ το κομμάτι της όψης είναι ένα σύνολο από στατικές σελίδες HTML και δυναμικές σελίδες Java Server Pages.

Τα πλενονεκτήματα του Struts συνοπτικά είναι τα ακόλουθα:

- Κεντρικές ρυθμίσεις μέσω αρχείων. Αντί για να ενσωματώνονται οι ρυθμίσεις στον κώδικα, το Struts προσφέρει διάφορα αρχεία ρυθμίσεων και XML κείμενα τα οποία επιτρέπουν στον προγραμματιστή την εύκολη αλλαγή των ρυθμίσεων χωρίς να χρειάζεται να μεταγλωττιστεί ξανά ο κώδικας της εφαρμογής.
- Beans για φόρμες. Πρόκειται για αντικείμενα JavaBeans τα οποία αποθηκεύουν τα δεδομένα από τις φόρμες της εφαρμογής. Με αυτόν τον τρόπο ο προγραμματιστής αποκτά ευκολότερη πρόσβαση στις παραμέτρους που εισάγει ο χρήστης.
- Bean Tags. Είναι ένα σύνολο από JSP tags που επιτρέπουν την εύκολη παρουσίαση των παραμέτρων των διάφορων JavaBeans που χρησιμοποιεί η εφαρμογή.
- HTML Tags. Επιπλέον σύνολο JSP tags που επιτρέπει την εύκολη δημιουργία HTML φορμών. Επιτρέπουν την ανάθεση τιμών σε πεδία της φόρμας από τις τιμές κάποιων πεδίων αντικειμένων ή την επανεμφάνιση της ίδιας φόρμας, με κάποιες τιμές να είναι συμπληρωμένες ανάλογα με την προηγούμενη είσοδο του χρήστη.
- Επικύρωση των πεδίων. Το Struts προσφέρει δυνατότητες για έλεγχο της εγκυρότητας των πεδίων μιας φόρμας, όπως π.χ. αν μια ημερομηνία είναι σωστή.

2.6.3 JAVA SERVER FACES (JSF)

Τα Java Server Faces [49] (JSF) είναι ένας συνδυασμός του Struts και του Swing [50]. Το JSF προσφέρει διαχείριση του κύκλου ζωής μιας εφαρμογής ιστού μέσω ενός servlet – ελεγκτή (όπως γίνεται και στο Struts), ενώ η διεπαφή με το χρήστη γίνεται μέσω συστατικών τα οποία μπορούν να χειρίζονται γεγονότα (όπως γίνεται στο Swing). Το μεγαλύτερο πλεονέκτημα των JSF είναι ότι ο προγραμματιστής μπορεί να αναπτύξει επαναχρησιμοποιήσιμα συστατικά για τις εφαρμογές ιστού του.

Οι δυνατότητες που προσφέρουν τα JSF στον προγραμματιστή είναι οι ακόλουθες:

- Προδιαγραφές για την περιήγηση στις σελίδες

- Στάνταρτ συστατικά για τη διεπιφάνεια του χρήστη, όπως πεδία εισόδου, κουμπιά και σύνδεσμοί
- Έλεγχος εγκυρότητας της εισόδου του χρήστη
- Εύκολος χειρισμός λαθών
- Διαχείριση Java bean
- Διαχείριση γεγονότων
- Υποστήριξη για πολυγλωσσικότητα.

2.6.4 PORTLETS

Όπως αναφέρεται στις προδιαγραφές των portlet [51], «ένα portlet είναι ένα δικτυακό συστατικό γραμμένο σε Java, το οποίο το διαχειρίζεται ένα portlet container, και το οποίο επεξεργάζεται αιτήσεις και δημιουργεί δυναμικό περιεχόμενο. Τα portlets χρησιμοποιούνται από δικτυακές πύλες σαν συστατικά της διεπαφής χρήστης που μπορούν να συνδεθούν δυναμικά και παρέχουν ένα στρώμα παρουσίασης για το πληροφοριακό σύστημα που βρίσκεται σε χαμηλότερο επίπεδο».

Στην πράξη, τα portlets είναι τμήματα κώδικα τα οποία είναι υπεύθυνα για τη λογική και την εμφάνιση ενός κομματιού μιας δικτυακής πύλης. Εγκαθίστανται σε portlet containers, τα οποία αποτελούν και το περιβάλλον εκτέλεσης τους. Όπως γίνεται φανερό, τα portlets και οι portlet containers μοιράζονται πολλές κοινές ιδέες με τα servlets και τα servlet containers.

Ένα portlet χειρίζεται αιτήσεις και δημιουργεί δυναμικό περιεχόμενο, το οποίο ονομάζεται τεμάχιο (fragment). Τα τεμάχια είναι σε κάποια γλώσσα markup, όπως HTML, XHTML ή WML. Τα τεμάχια από τα διάφορα portlets συνενώνονται σε ένα ολοκληρωμένο κείμενο.

Το πρόγραμμα – πελάτης του χρήστη αλληλεπιδρά με τα portlets στέλνοντας αιτήσεις και λαμβάνοντας κάποια απάντηση. Αυτή η αλληλεπίδραση μπορεί να είναι κάποιος σύνδεσμός που θα ακολουθήσει ο χρήστης ή η αποστολή δεδομένων χρησιμοποιώντας μια φόρμα. Οι ενέργειες του χρήστη προωθούνται από το portlet container στο κατάλληλο portlet το οποίο αναλαμβάνει να εκτελέσει τις κατάλληλες ενέργειες.

Όπως αναφέρθηκε παραπάνω, τα portlets μοιράζονται πολλές ιδέες με τα servlets. Όπως γίνεται και με το servlet container και τα servlets, έτσι και τον κύκλος ζωής των portlet τον διαχειρίζεται ο portlet container:

- Init. Αρχικοποίηση του portlet.
- Διαχείριση αιτήσεων. Επεξεργασία των ενεργειών του χρήστη.

- Καταστροφή. Λήξη του κύκλου ζωής του portlet.

Μία επιπλέον δυνατότητα των portlets είναι ότι έχουν καταστάσεις λειτουργίας οι οποίες δείχνουν τη λειτουργία που εκτελεί το portlet μία δεδομένη χρονική στιγμή. Οι καταστάσεις λειτουργίας μπορούν να χωριστούν σε τρείς κύριες κατηγορίες:

- Υποχρεωτικές καταστάσεις. Αυτές είναι οι καταστάσεις Επεξεργασίας (Edit), Βοήθειας (Help) και Όψης (View). Ένα portlet πρέπει να υποστηρίζει την κατάσταση Όψης προκειμένου να μπορεί να δημιουργήσει το κατάλληλο markup για μία σελίδα. Η κατάσταση Επεξεργασίας επιτρέπει στο χρήστη να μεταβάλλει τα δεδομένα που χρησιμοποιεί το portlet, ενώ ή κατάσταση Βοήθειας εμφανίζει μια οθόνη με βοήθεια για το χρήστη.
- Προαιρετικές ειδικές καταστάσεις. Τέτοιες καταστάσεις περιλαμβάνουν την κατάσταση Σχετικά, (About) όπου εμφανίζονται πληροφορίες για το portlet, η κατάσταση Ρυθμίσεων (Config) που επιτρέπει στους διαχειριστές να αλλάξουν τις ρυθμίσεις ενός portlet, Επεξεργασία_Αρχικών_Ρυθμίσεων (Edit_defaults) που επιτρέπει στους διαχειριστές να ορίσουν εκ των προτέρων κάποιες τιμές για την κατάσταση Επεξεργασίας, και η κατάσταση Εκτύπωσης (Print) για την εμφάνιση του περιεχομένου ενός portlet σε μορφή που μπορεί να εκτυπωθεί εύκολα.
- Καταστάσεις που εξαρτώνται από τον κατασκευαστή της δικτυακής πύλης. Οι καταστάσεις αυτές δεν ορίζονται στις προδιαγραφές και εξαρτώνται από τον κατασκευαστή της δικτυακής πύλης.

Τέλος, μία ακόμα σημαντική ιδιότητα των portlets είναι ότι μπορούν να λειτουργούν σαν «παράθυρα» μέσα σε μια σελίδα. Με αυτό τον τρόπο ο χρήστης μπορεί να παραμετροποιήσει τον τρόπο εμφάνισης της κάθε σελίδας, να μεγιστοποιήσει ή να ελαχιστοποιήσει το μέγεθος ενός portlet κλπ.

2.6.5 RUBY ON RAILS

Το Ruby on Rails [52] (RoR) είναι ένα περιβάλλον ανάπτυξης δικτυακών εφαρμογών που βασίζεται στη γλώσσα προγραμματισμού Ruby. Το μεγαλύτερο πλεονέκτημα του RoR's προέρχεται από τη Ruby. Η Ruby επιτρέπει την εύκολη δημιουργία γλωσσών για συγκεκριμένο αντικείμενο (domain-specific languages) και μετα-προγραμματισμό. Τα πλεονεκτήματα που προσφέρει το RoR είναι:

- Πλήρης στοίβα MVC. Συγκριτικά με τις αντίστοιχες λύσεις σε Java, το RoR είναι μια πλατφόρμα ανάπτυξης που βασίζεται στο σχεδιαστικό πρότυπο MVC και παρέχει όλα τα απαραίτητα επίπεδα. Μερικές λύσεις σε Java απαιτούν το

συνδυασμό πολλών διαφορετικών πλατφόρμων ανάπτυξης προκειμένου να υπάρχει ολόκληρη η στοίβα (για παράδειγμα συνδυασμός των Struts, Hibernate και Tiles).

- Δεν υπάρχουν αρχεία ρυθμίσεων XML. Το RoR χρησιμοποιεί προηγμένους μηχανισμούς ανακάλυψης (discovery) και απεικόνισης (reflection)• για παράδειγμα η απεικόνιση χρησιμοποιείται για την αντιστοίχιση πινάκων μίας βάσης δεδομένων σε αντικείμενα της Ruby.
- Λιγότερος κώδικας. Οι μηχανισμοί απεικόνισης και η ανακάλυψης αναλαμβάνουν λεπτομέρειες χαμηλότερου επιπέδου και δημιουργούν αυτόματα τον απαραίτητο κώδικα.
- Μικρός χρόνος ανάπτυξης. Η κλασική μέθοδος δοκιμής μιας αλλαγής σε μία εφαρμογή ιστού περιλαμβάνει την εισαγωγή των ρυθμίσεων, τη μεταγλώττιση, την εγκατάσταση, την επανεκκίνηση και τη δοκιμή της εφαρμογής, μια αρκετά χρονοβόρα διαδικασία. Στο RoR, οι αλλαγές λειτουργούν τη στιγμή που αυτές πραγματοποιούνται.
- Scaffolding. Το RoR δημιουργεί αυτόματα διαδικασίες για δημιουργία, ανάκτηση, ενημέρωση και διαγραφή δεδομένων από κάθε πίνακα της βάσης δεδομένων, καθώς και όψεις για αυτά.

2.6.6 WEBFORMS

Το WebForms είναι η πρόταση της Microsoft για την ανάπτυξη εφαρμογών ιστού με μια πιο δομημένη μέθοδο. Η βασική ιδέα πίσω από το WebForms είναι να αντιγραφεί το μοντέλο που χρησιμοποιείται για την ανάπτυξη «πλούσιων» εφαρμογών σε περιβάλλον Internet. Προκειμένου να πραγματοποιηθεί αυτό, χρησιμοποιήθηκε η τεχνολογία ASP.NET [53]. Τα σημαντικότερα πλεονεκτήματα του WebForms είναι:

- Καθαρός διαχωρισμός ανάμεσα στη λογική της εφαρμογής (κώδικας στην πλευρά του εξυπηρετητή) και το στρώμα παρουσίασης (HTML).
- Ένα μεγάλο σύνολο από συστατικά τα οποία αυτόματα αποδίδονται σε HTML κατάλληλο για κάθε εφαρμογή-πελάτη και δυνατότητα διαχείρισης της κατάστασής τους.
- Βελτιωμένη διαχείριση κατάστασης της εφαρμογής (session management).
- Προγραμματιστικό μοντέλο που βασίζεται σε γεγονότα στην πλευρά του εξυπηρετητή.

- Η λογική της εφαρμογής μπορεί να γραφτεί σε οποιαδήποτε γλώσσα του Microsoft .NET VB, C#, Managed C++ κλπ).
- Η χρησιμοποίηση του Visual Studio .NET σαν εργαλείο ανάπτυξης, γεγονός που απλοποιεί τη διαδικασία ανάπτυξης της εφαρμογής.

2.7 CATALYST

Το Catalyst [54] είναι μια πλατφόρμα ανάπτυξης εφαρμογών ιστού βασισμένο στη γλώσσα προγραμματισμού Perl. Χρησιμοποιεί και αυτό το σχεδιαστικό πρότυπο MVC. Η φιλοσοφία που υπάρχει στην ομάδα ανάπτυξης του Catalyst είναι να υπάρχει εύκολη πρόσβαση στα εργαλεία που ο προγραμματιστής χρειάζεται για την ανάπτυξη της εφαρμογής του, επιβάλλοντας μικρούς περιορισμούς στον τρόπο που ο προγραμματιστής θα χρησιμοποιήσει αυτά τα εργαλεία.

Το σημαντικότερο πλεονέκτημα του Catalyst είναι η μεγάλη βιβλιοθήκη κώδικα που διαθέτει, η CPAN [55] και η δυνατότητα ταυτόχρονης υποστήριξης πολλών μοντέλων, ελεγκτών και όψεων μέσα στην ίδια εφαρμογή. Για παράδειγμα το τελικό κείμενο HTML μπορεί να προκύψει με πολλούς διαφορετικούς τρόπους, χρησιμοποιώντας διάφορους «οδηγούς» όπως Template Toolkit, Mason, HTML::Template ή ακόμα και PHP. Μία ακόμα σημαντική δυνατότητα είναι ότι υποστηρίζει μεγάλο αριθμό βάσεων δεδομένων, με υποστήριξη για «εικονικές» βάσεις δεδομένων, όπως το Google, το Amazon και αρχεία CSV.

2.7.1 ZOPE

Το Zope [56] είναι ένας αντικειμενοστραφής εξυπηρετητής για εφαρμογές ιστού γραμμένος σε Python. Η κεντρική ιδέα του Zope είναι ότι η κάθε εφαρμογή αποτελείται από αντικείμενα και διευθύνσεις (URL) οι οποίες «δείχνουν» σε αντικείμενα. Αυτή η αντικειμενοστραφής προσέγγιση επιτρέπει στον προγραμματιστή να χρησιμοποιήσει τεχνικές αντικειμενοστραφούς προγραμματισμούς όπως π.χ. η ενθυλάκωση. Τα αντικείμενα ταξινομούνται σε μια ιεραρχική δομή. Όταν ο χρήστης προσπαθεί να δει μια συγκεκριμένη διεύθυνση, ο εξυπηρετητής του Zope αντιστοιχεί ένα αντικείμενο στο URL μέσω μιας ιεραρχίας η οποία περιγράφει ποιο αντικείμενο περιέχει ποια άλλα αντικείμενα (containment hierarchy). Η συγκεκριμένη ιεραρχία αποτελεί ένα από τις πιο σημαντικότερα πλεονεκτήματα του Zope. Το δεύτερο σημαντικό πλεονέκτημα είναι η κληρονομικότητα μέσω της παραπάνω ιεραρχίας. Η δυνατότητα αυτή ονομάζεται acquisition και επιτρέπει σε

ένα αντικείμενο να κληρονομεί τις ιδιότητες του αντικειμένου το οποίο το περιέχει, με τον ίδιο ακριβώς τρόπο που μία κλάση κληρονομεί πεδία και μεθόδους από την υπερκλάση της.

3 ΚΙΝΗΤΟΙ ΠΡΑΚΤΟΡΕΣ

Σε αυτή την ενότητα παρουσιάζουμε τη σχεδίαση και υλοποίηση μιας διαδικτυακής πλατφόρμας κινητών πρακτόρων βασισμένης στις υπηρεσίες ιστού. Ο τρόπος σχεδίασης και υλοποίησης της πλατφόρμας αποδεικνύεται ότι είναι αρκετά ευέλικτος και παρουσιάζει μια σειρά από πλεονεκτήματα τόσο από πλευράς εύκολης ανάπτυξης της πλατφόρμας όσο και από πλευράς λειτουργικότητας και επεκτασιμότητας.
Οι συνιστάμενες οντότητες της πλατφόρμας εγκαθίστανται σε διακομιστές Apache Tomcat, ενώ η αρχική υλοποίηση βασίστηκε στην βιβλιοθήκη Apache SOAP.

3.1 ΕΙΣΑΓΩΓΗ

Η χρήση των υπηρεσιών ιστού στον διαδίκτυο αναπτύσσεται ολοένα με γρηγορότερο ρυθμό καθώς η ανάγκη για επικοινωνία σε επίπεδο εφαρμογών και διαλειτουργικότητα αυξάνει στα συστήματα κατανεμημένης αρχιτεκτονικής.
Το προγραμματιστικό μοντέλο των πρακτόρων εισάγει ένα σημαντικό παράδειγμα για την δημιουργία κατανεμημένων εφαρμογών σε ανοικτά και δυναμικά μεταβαλλόμενα υπολογιστικά περιβάλλοντα. Μια ενδιαφέρουσα περιοχή εφαρμογής των κινητών πρακτόρων είναι το διαδίκτυο. Οι κινητοί πράκτορες στηριζόμενοι στην ιδέα του κινητού κώδικα [9] και αποτελούν μια ευέλικτη και δυναμική δομή ικανή να μεταναστεύει από κάποιο διακομιστή σε ένα άλλο και να αλληλεπιδρά τοπικά με αυτούς.
Επομένως, σε ένα περιβάλλον ολοκλήρωσης του διαδικτύου και των πρακτόρων, οι κινητοί πράκτορες μπορούν να εκκινηθούν από ένα διαδικτυακό τόπο σε ένα άλλο, πραγματοποιώντας συναλλαγές βασιζόμενοι στην λογική της εφαρμογής. Αυτό το σενάριο είναι ιδιαίτερα ελκυστικό σήμερα με τις ανάπτυξη των ασύρματων κινητών συσκευών όπου ένα χρήστης θα μπορούσε για παράδειγμα να εκκινήσει ένα ειδικό κινητό πράκτορα για να συλλέξει πληροφορία στο διαδίκτυο ή να πραγματοποιήσει μια συναλλαγή ηλεκτρονικού εμπορίου (e-commerce), να κλείσει την ασύρματη συσκευή του και να επασυνδεθεί αργότερα για να δει τα αποτελέσματα της αναζήτησης ή της συναλλαγής του κινητού πράκτορα.
Υπάρχοντα συστήματα που παρέχουν ολοκλήρωση μεταξύ των κινητών πρακτόρων και διακομιστών διαδικτύου ακολουθούν είτε την προσέγγιση μιας πλατφόρμας κινητών πρακτόρων που είναι λίγο ή καθόλου διασυνδεδεμένη με τον διακομιστή [57], [58] ή υλοποιούν ειδικούς διακομιστές ικανούς εκτός των άλλων να φιλοξενήσουν πράκτορες

λογισμικού [59]. Άλλες επίσης ερευνητικές προσπάθειες [60], [61] προτείνουν διαφορετικούς μηχανισμούς για την στενή ολοκλήρωση πλατφόρμων κινητών πρακτόρων σε ήδη υπάρχουσες υποδομές από διακομιστές διαδικτύου βασιζόμενες κυρίως στην τεχνολογία των Servlets [62] και το πρότυπο JavaBeans [63]. Επιπλέον στην αναφορά [64] προτείνεται μια γλώσσα σεναρίου (scripting language) πρακτόρων βασισμένη στην XML.

Σε αντίθεση με τα παραπάνω η υλοποίηση που παρουσιάζουμε σε αυτή την ενότητα βασίζεται στο μοντέλο των υπηρεσιών ιστού και ειδικά στο πρωτόκολλο SOAP [10] όπως αυτό χρησιμοποιείται στον παγκόσμιο ιστό.

Η προσέγγιση αυτή ξεκινά από ήδη διαθέσιμες υποδομές διότι προσδίδει τις λειτουργίες των κινητών πρακτόρων σε ήδη υπάρχοντες διακομιστές διαδικτύου.

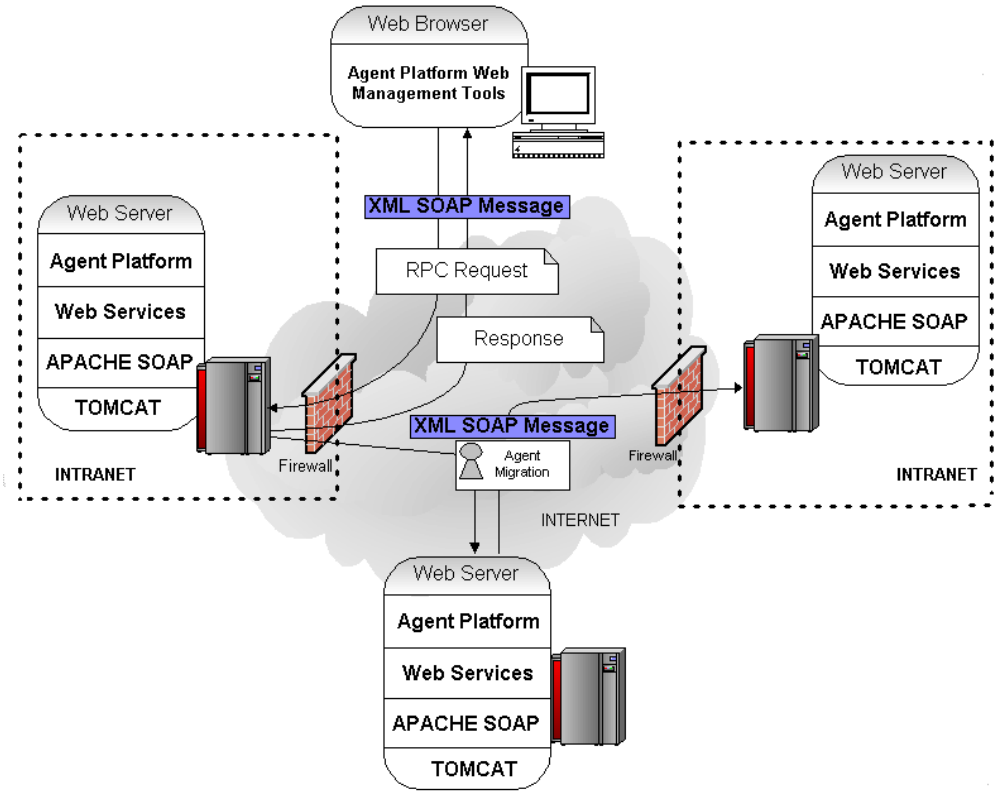
Επιπρόσθετα μας επιτρέπει να εκμεταλλευτούμε την μεγάλη έρευνα που επικεντρώνεται στις υπηρεσίες ιστού, στο SOAP και στην γλώσσα XML και καθιστά δυνατή όχι μόνο την εύκολη πρόσβαση σε υπηρεσίες πρακτόρων αλλά και μια εύκολη υλοποίηση μιας πλατφόρμας διαχείρισης ως ένα σύνολο από υπηρεσίες ιστού.

Τα συστατικά της πλατφόρμας αναπτύσσονται σε διακομιστές διαδικτύου Apache [65] χρησιμοποιώντας την βιβλιοθήκη Apache SOAP [66] και αποτελείται από ένα σύνολο προγραμματιστικών διεπαφών (API) και ένα σύνολο από JSP σελίδες για την διαχείριση της.

Η πλατφόρμα σχεδιάστηκε με τέτοιο τρόπο ώστε να είναι προσαρμόσιμη και επεκτάσιμη. Για την επίτευξη αυτού του σκοπού εισάγαμε ένα επίπεδο μεταξύ του συστήματος διαχείρισης των πρακτόρων και του καναλιού επικοινωνίας ώστε να ενθυλακώσουμε τις κλήσεις τις σχετιζόμενες με το SOAP παρέχοντας ένα απλό σχετικά σύνολο προγραμματιστικών διεπαφών (API) στον χρήστη της πλατφόρμας.

3.2 ΑΡΧΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ ΚΙΝΗΤΩΝ ΠΡΑΚΤΟΡΩΝ

Προκειμένου να επιτύχουμε την ολοκλήρωση των κινητών πρακτόρων σε εξυπηρετητές Ιστού, καταλήξαμε στην αρχιτεκτονική που απεικονίζεται στην Εικόνα 3.1.



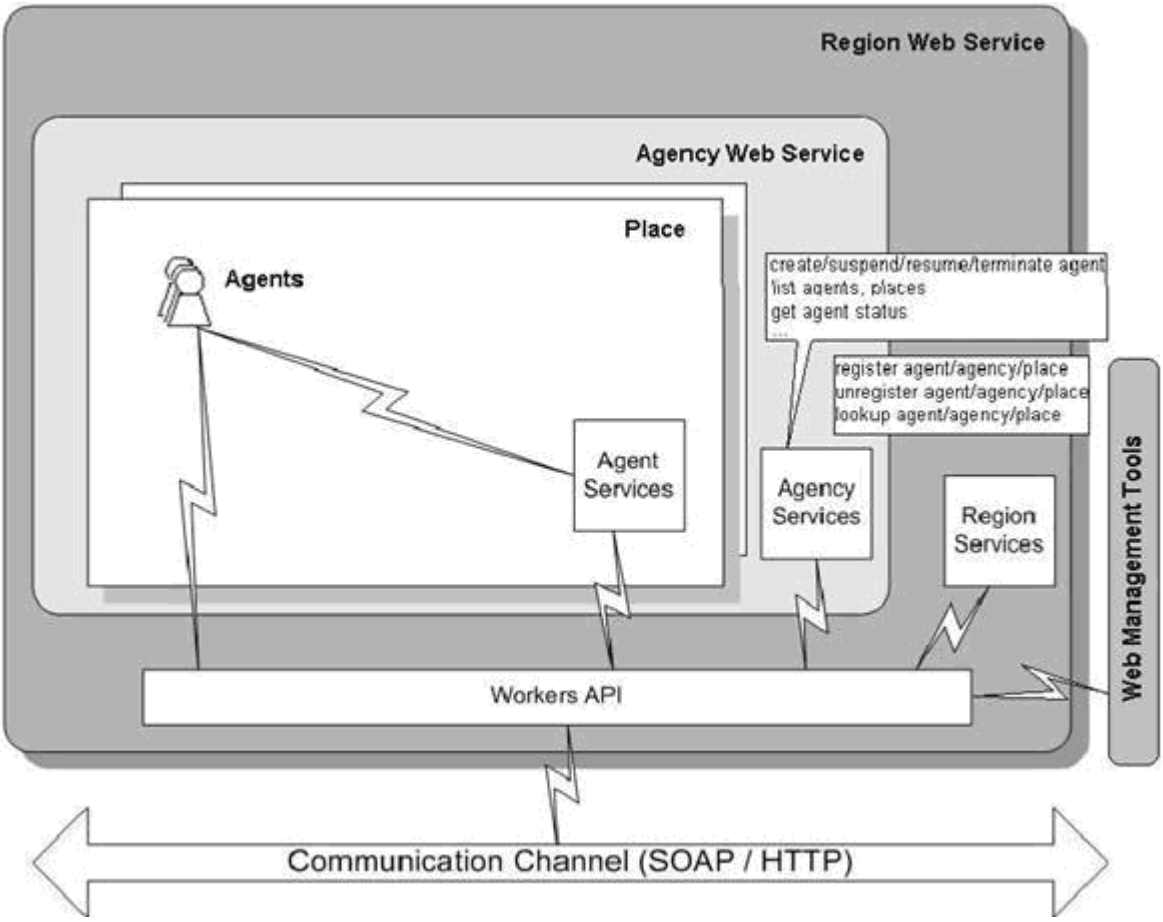
Εικόνα 3.1: Η γενική αρχιτεκτονική της πλατφόρμας κινητών πρακτόρων SOAP

Οι συνιστώσες της πλατφόρμας υλοποιήθηκαν εξ' ολοκλήρου χρησιμοποιώντας τη γλώσσα Java και αποτελούνται τον κύριο πυρήνα συνιστωσών στην πλευρά του εξυπηρετητή και από τις διαχειριστικά εργαλεία στην πλευρά του πελάτη. Όλες οι συνιστώσες λογισμικού εγκαθίστανται σε TOMCAT υποδοχείς servlets. Η πλατφόρμα εκμεταλλεύεται τις κλήσεις SOAP-RPC χρησιμοποιώντας της βιβλιοθήκη Apache SOAP. Το SOAP επιτρέπει σε επίσης την εγκατάσταση της πλατφόρμας πρακτόρων στα δίκτυα που ασφαλίζονται από διακομιστές firewall. Η πλατφόρμα πρακτόρων αποτελείται από δύο μέρη, την πλευρά εξυπηρετητή και την πλευρά πελάτη.

3.2.1 Η ΠΛΕΥΡΑ ΤΟΥ ΕΞΥΠΗΡΕΤΗΤΗ

Η πλευρά του εξυπηρετητή περιέχει τα κύρια συστατικά της πλατφόρμας. Όπως φαίνεται στην Εικόνα 3.2, έχουμε υιοθετήσει τις έννοιες των προδιαγραφών MASIF [68], των θέσεων (places), των πρακτορείων (agencies) και των περιοχών (regions) που χρησιμοποιούνται επίσης σε διάφορες υπάρχουσες πλατφόρμες πρακτόρων [69]. Η βασική διαφορά είναι ότι τώρα το πρακτορείο και η περιοχή είναι υπηρεσίες Ιστού που εκθέτουν τις μεθόδους σε άλλες συνιστώσες λογισμικού της πλατφόρμας.

Κάθε διακομιστής ιστού περιέχει δύο κύρια αντικείμενα, το πρακτορείο και την περιοχή με την ύπαρξη της τελευταίας να μην είναι απαραίτητη. Η τρέχουσα υλοποίηση της πλατφόρμας επιτρέπει μόνο ένα πρακτορείο σε κάθε διακομιστή ιστού.



Εικόνα 3.2: Οι λογικές συνιστώσες της πλατφόρμας των πρακτόρων

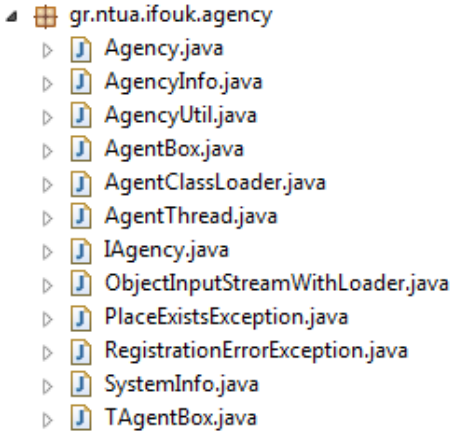
Η υπηρεσία ιστού "περιοχή" χρησιμοποιείται ως υπηρεσία καταλόγου. Η δημιουργία, μετακίνηση καθώς και κάθε κύρια λειτουργία ενός πράκτορα κατά τον κύκλο ζωής του έχει ως αποτέλεσμα την ενημέρωση της περιοχής από το πρακτορείο που έχει καταχωρηθεί στα μητρώα του. Επιπρόσθετα ένα πρακτορείο ή ένας πράκτορας (διαμέσου του πρακτορείου) μπορούν να αναζητήσουν πληροφορία από την περιοχή για την διαθεσιμότητα άλλων πρακτορείων ή τη θέση άλλων πρακτόρων κ.τ.λ. με χρήση ειδικών φίλτρων αναζήτησης.

Οι υπηρεσίες ιστού "πρακτορεία" αποτελούν τις κύριες συνιστώσες λογισμικού της πλατφόρμας, παρέχοντας το περιβάλλον εκτέλεσης και τη δίοδο επικοινωνίας για άλλα πρακτορεία και πράκτορες. Μια υπηρεσία ιστού "πρακτορείο" είναι ικανή να δεχτεί είτε κινητούς είτε στατικούς πράκτορες. Ο κύριος σκοπός της είναι η δημιουργία, αποθήκευση, μετακίνηση πρακτόρων και στο παρέχει υπηρεσίας επικοινωνίας μεταξύ τους. Κάθε

πρακτορείο χωρίζεται σε τοποθεσίες (places). Κάθε τοποθεσία ομαδοποιεί τη λειτουργικότητα στο πρακτορείο ενθυλακώνοντας ορισμένες δυνατότητες και περιορισμούς σε επισκεπτόμενους πράκτορες , βοηθώντας έτσι τον προγραμματιστή να διαχωρίσει τους πράκτορες μεταξύ τους.

Τα κυριότερα συστατικά της πλατφόρμας, καθώς και οι κλάσεις και οι διεπαφές που τα υλοποιούν περιγράφονται στις επόμενες σελίδες.

Πρακτορείο



Διεπαφή IAgency: Ορίζει τις βασικές λειτουργίες που πρέπει να υλοποιεί ένα πρακτορείο.

Κλάση Agency: Η κλάση αυτή αποτελεί την καρδιά της υπηρεσίας Ιστού "Agency". Η κλάση αυτή στιγμιοτυπείται από τη μηχανή SOAP ως εφαρμογή (ορίζεται στο περιγραφικό αρχείο εγκατάστασης της υπηρεσίας) και επομένως σε ένα εξυπηρετητή Ιστού μόνο ένα νήμα εκτέλεσης αυτής της κλάσης είναι ενεργό. Οι βασικές λειτουργίες των μεθόδων της υπηρεσίας είναι η διαχείριση των δομών των τοποθεσιών και πρακτόρων τους οποίους κρατά σε δομές τύπου Hashtable. Επομένως παρέχει μεθόδους δημιουργίας, εύρεσης και διαγραφής πρακτόρων και τοποθεσιών. Οι μέθοδοι αυτές περιλαμβάνουν ενημέρωση της περιοχής όπου χρειάζεται. Οι πιο πολύπλοκες μέθοδοι αφορούν την μετανάστευση πρακτόρων. Αυτές είναι η requestTransfer και η retrieveAgent. Η μέθοδος requestTransfer καλείται από τον ενθυλακωτή ενός κινητού πράκτορα όταν ο προγραμματιστής καλέσει τη μέθοδο move με παραμέτρους το απομακρυσμένο πρακτορείο και την τοποθεσία στην οποία θέλει να μεταναστεύσει ο κινητός του πράκτορας

Κλάση AgencyInfo: Περιέχει πληροφορίες για το πρακτορείο όπως η διεύθυνσή, το όνομά του και η περιοχή στην οποία είναι εγγεγραμμένο.

Κλάση AgencyUtil: Περιέχει χρήσιμες λειτουργίες

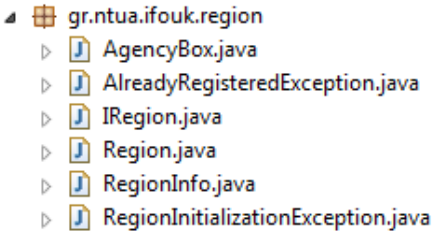
Κλάση AgentClassLoader: Η κλάση αυτή αναλαμβάνει τη φόρτωση των κλάσεων που αποτελούν τον κάθε πράκτορα. Οι κλάσεις που αποτελούν τον κάθε πράκτορα δεν βρίσκονται από την αρχή στο κάθε πρακτορείο, αλλά μεταφέρονται σε αυτό όταν δημιουργείται ένας καινούργιος πράκτορας ή μετακινείται κάποιος από άλλο πρακτορείο. Χρησιμοποιώντας τις δυνατότητες της Java για reflection το πρακτορείο φορτώνει δυναμικά στον χρόνο εκτέλεσης τις νέες αυτές κλάσεις.

Κλάση AgentThread: Είναι το νήμα (thread) το οποίο αναλαμβάνει τη διαχείριση του κύκλου ζωής του κάθε πράκτορα.

Κλάση AgentBox: Κλάση που ενθυλακώνει όλα τα αντικείμενα που σχετίζονται με τον κάθε πράκτορα, όπως είναι το νήμα εκτέλεσης ο φορτωτής κλάσεων, το αναγνωριστικό και τις πληροφορίες του πράκτορα.

Κλάση TAgentBox: Ενθυλακώνει ένα υποσύνολο των αντικειμένων που περιγράφονται στην κλάση AgentBox και οι οποίες μεταφέρονται μαζί με τον πράκτορα όταν αυτός μετακινείται από μία τοποθεσία σε μία άλλη.

Περιοχή



Διεπαφή IRegion: Ορίζει τις βασικές λειτουργίες που πρέπει να υλοποιεί μία περιοχή.

Κλάση Region: Υλοποιεί την διεπαφή IRegion και τη λογική της περιοχής. Όπως και η κλάση Agency που περιγράφηκε παραπάνω στιγμιοτυπείται από τη μηχανή SOAP σαν ανεξάρτητη εφαρμογή στο δικό της νήμα εκτέλεσης. Περιέχει τις κατάλληλες λειτουργίες που επιτρέπουν σε απομακρυσμένα πρακτορεία να εγγραφούν και να ενημερώσουν για αλλαγές στην κατάστασή τους (π.χ. προσθήκη μέρους ή κινητού πράκτορα), καθώς και να γίνει αναζήτηση για επιμέρους οντότητες της πλατφόρμας όπως είναι τα πρακτορεία και οι πράκτορες.

Κλάση RegionInfo: Περιέχει πληροφορίες για την περιοχή όπως η διεύθυνσή της και το όνομά της.

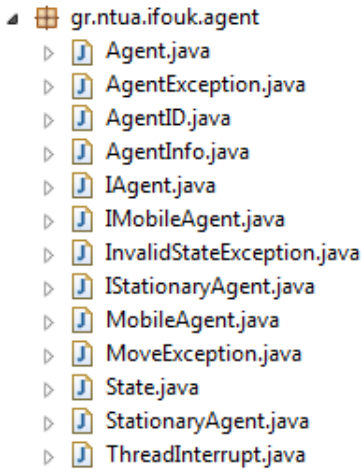
Κλάση AgencyBox: Χρησιμοποιείται για την αποθήκευση πληροφοριών για κάθε πρακτορείο που έχει εγγραφεί στην περιοχή όπως είναι η διεύθυνσή του, το όνομά του και η λίστα με τα μέρη που αυτό περιέχει.

Πράκτορες

Στην Εικόνα 3.3 παρουσιάζεται η ιεραρχία των κλάσεων των πρακτόρων στη γλώσσα UML (Unified Modelling Language).

Οι κλάσεις και διεπαφές που αναπτύχθηκαν είναι οι ακόλουθες:

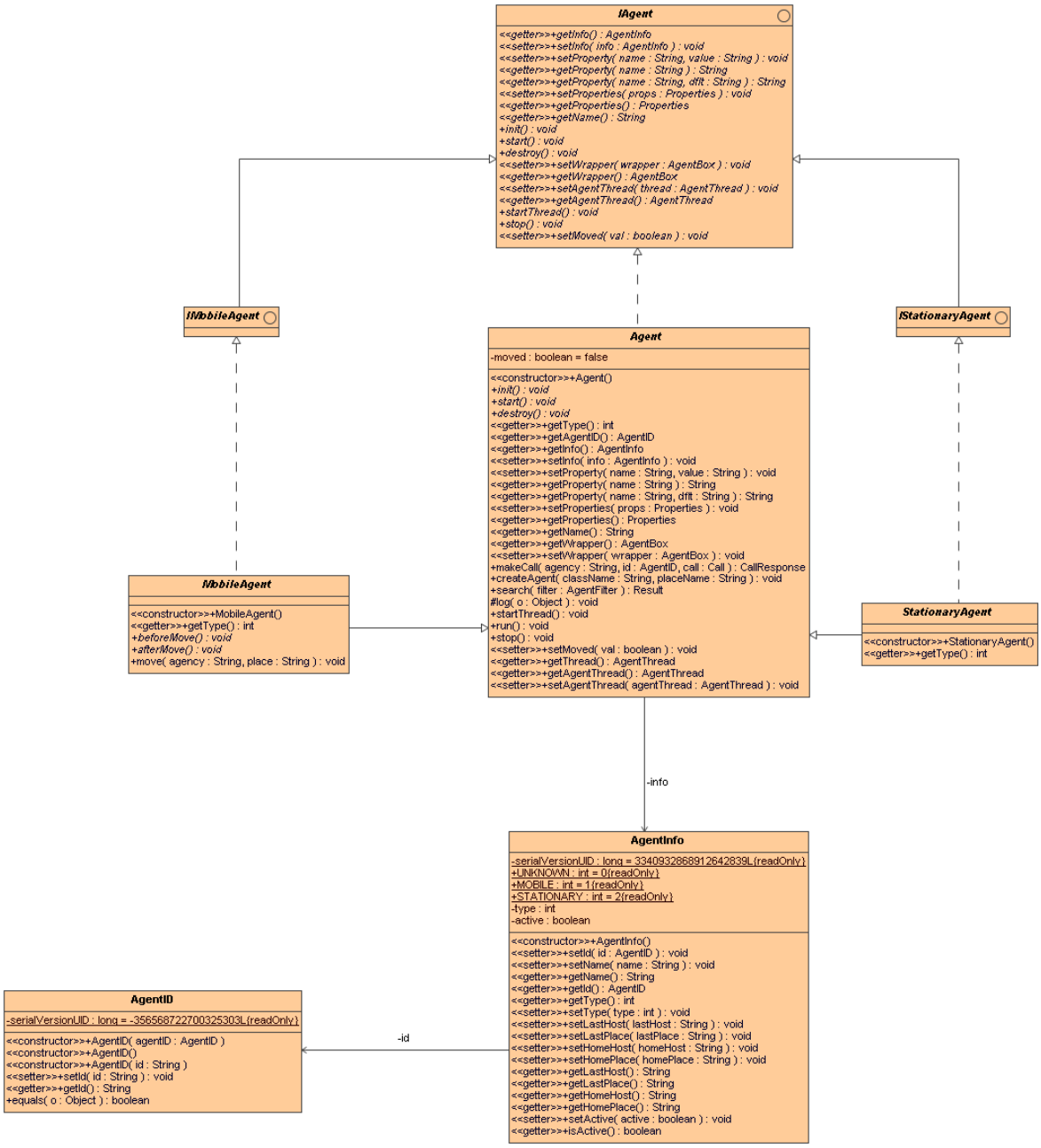
Διεπαφή IAgent: Η διεπαφή αυτή ορίζει την υπογραφή των μεθόδων της κλάσης γονέα Agent, καθώς και τον κύκλο ζωής κάθε πράκτορα. Η διεπαφή αυτή επεκτείνεται από τις διεπαφές IMobileAgent και IStationaryAgent οι οποίες διαχωρίζουν την λειτουργικότητα που παρέχουν οι κινητοί και στατικοί πράκτορες.



Κλάση Agent: Η κλάση αυτή παρέχει μια υλοποίησης πρώτου επιπέδου της διεπαφής IAgent, η οποία προδιαγράφει τις μεθόδους ενός αντικειμένου πράκτορα που μπορεί να καλέσει η κλάση ενθυλάκωσης AgentBox. Ουσιαστικά δηλαδή υλοποιεί τις κοινές μεθόδους των πρακτόρων (είτε είναι κινητοί είτε όχι), οι οποίες καλούνται από το περιβάλλον εκτέλεσης (agent execution environment), δηλαδή από τις υπηρεσίες των πρακτορείων.

Διεπαφή IMobileAgent: Ορίζει τις επιπλέον μεθόδους που πρέπει να υλοποιεί ένας κινητός πράκτορας.

Κλάση MobileAgent: Η abstract κλάση αυτή που αποτελεί την κλάση γονέα κάθε κινητού πράκτορα πρέπει να επεκταθεί από τον προγραμματιστή μιας εφαρμογής ώστε να υλοποιήσει τη λειτουργικότητα που επιθυμεί. Οι μέθοδοι που πρέπει να υλοποιηθούν είναι οι beforeMove, move και afterMove οι οποίες αναφέρονται αποκλειστικά σε κινητούς πράκτορες



Εικόνα 3.3: Διάγραμμα κλάσεων των πρακτόρων

Διεπαφή IStationaryAgent: Παρόλο που δεν περιέχει τον ορισμό κάποιων μεθόδων, η διεπαφή αυτή ορίζει τους στατικούς πράκτορες (marker interface).

Κλάση StationaryAgent: Η abstract κλάση αυτή που αποτελεί την κλάση γονέα κάθε στατικού πράκτορα και πρέπει να επεκταθεί από τον προγραμματιστή μιας εφαρμογής ώστε να υλοποιήσει τη λειτουργικότητα που επιθυμεί. Δεν απαιτείται η υλοποίηση κάποιων επιπλέον μεθόδων.

Κλάση AgentID: Η κλάση αυτή αναπαριστά ένα μοναδικό αναγνωριστικό (unique identifier) για κάθε κινητό πράκτορα προκειμένου να είναι δυνατή η αναγνώριση του κάθε πράκτορα τόσο από άλλους πράκτορες αλλά και από άλλα συστατικά της πλατφόρμας.

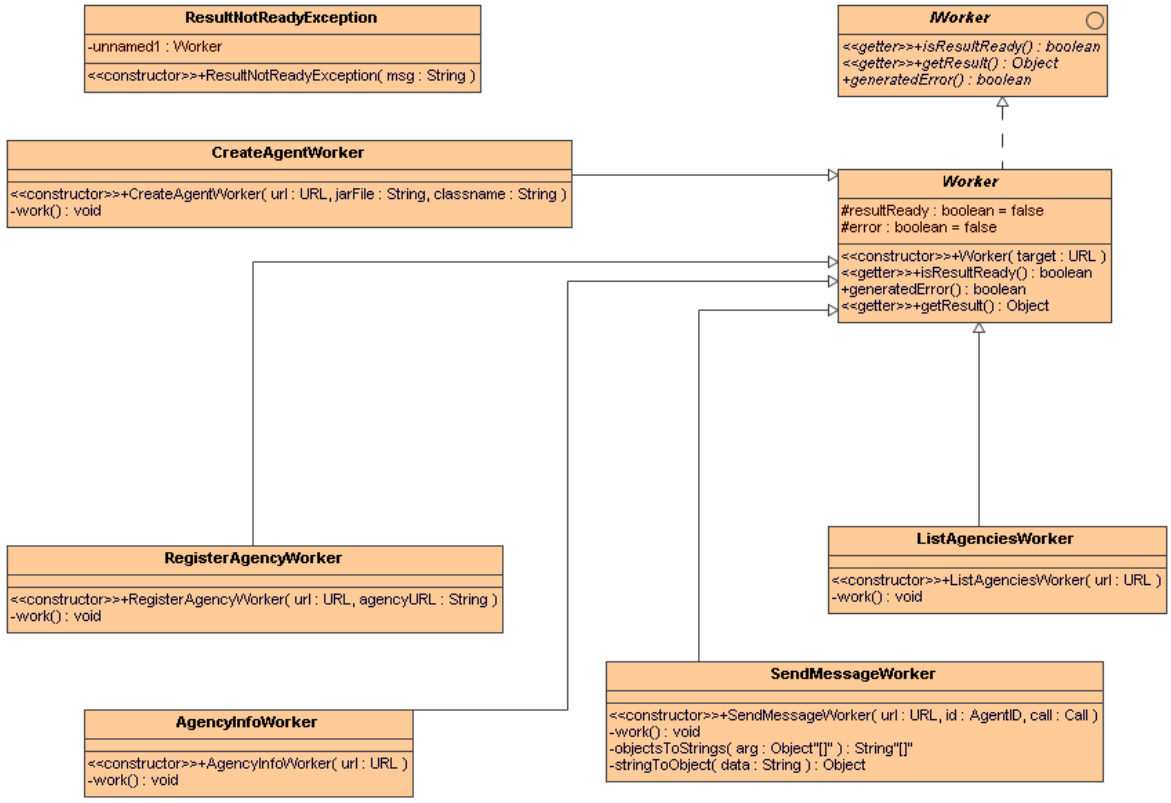
Κλάση AgentInfo: Χρησιμεύει για την αποθήκευση πληροφοριών όπως τη διεύθυνση του πρακτορείου που δημιουργήθηκε ο πράκτορας, το τελευταίο πρακτορείο πριν μεταβεί σε αυτό που βρίσκεται τώρα, το όνομα του κλπ.

Επικοινωνία ανάμεσα στα συστατικά της πλατφόρμας

Ο στόχος κατά το σχεδιασμό της πλατφόρμας ήταν οι συνιστώσες του λογισμικού της να είναι όσο το δυνατό μικρές και ευέλικτες και ταυτόχρονα να παρέχουν ένα μηχανισμό επεκτασιμότητας που θα επιτρέπει τη διασύνδεση επιπλέον λειτουργικότητας. Για το λόγο αυτό εισάγαμε το Workers API ως σύνδεσμο μεταξύ της πλατφόρμας και του επικοινωνιακού καναλιού. Οι Workers είναι απλές κλάσεις που είναι υπεύθυνες για ορισμένες στοιχειώδεις εργασίες όπως π.χ. τη δημιουργία μιας νέας τοποθεσίας σε ένα πρακτορείο, την ειδοποίηση της περιοχής για τυχόν αλλαγές στην κατάσταση ενός πράκτορα κ.τ.λ. Κάθε κλήση από κάποιο Worker απευθύνεται στο rpcrouter servlet το οποίο είναι υπεύθυνο να καλεί τις υπηρεσίες της πλατφόρμας ανάλογα με τις αιτήσεις που δέχεται. Οι αιτήσεις αυτές είναι σε XML μορφή (ο φάκελος SOAP) και το rpcrouter servlet είναι μετασχηματίζει αυτές τις αιτήσεις σε κλήσεις μεθόδων στις κατάλληλες υπηρεσίες ιστού.

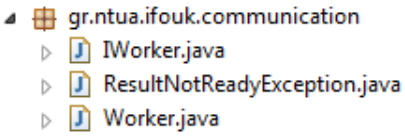
Υπάρχουν δύο κατηγορίες workers: αυτοί που καθιστούν δυνατή την επικοινωνία μεταξύ πρακτορείων και αυτοί που επιτρέπουν την επικοινωνία μεταξύ πρακτορείων και περιοχών. Για να είναι σε θέση να εκτελέσει τη συγκεκριμένη ενέργειά του, σε κάθε εργαζόμενο πρέπει να δοθούν διάφορες παράμετροι. Από τη χρησιμοποίηση αυτών των παραμέτρων, διαμορφώνει το αίτημα SOAP, κατόπιν προς τα εμπρός αυτό στη διεύθυνση παραληπτών και ανακτά το αποτέλεσμα, ενδεχομένως. Οι workers είναι ένα σημαντικό μέρος της πλατφόρμας. Αυτό το API καθιστά την επικοινωνία μεταξύ των απομακρυσμένων αντικειμένων διαφανή, επιτρέποντας κατά συνέπεια την εύκολη προσαρμογή της πλατφόρμας ώστε να χρησιμοποιεί άλλα κανάλια επικοινωνίας (όπως το RMI, τις υποδοχές και CORBA). Ένα άλλο σημαντικό πλεονέκτημα των Workers είναι ότι η ύπαρξή της μεγιστοποιεί την ικανότητα επαναχρησιμοποίησης κώδικα. Η πλευρά του χρήστη χρησιμοποιεί αυτό το API προκειμένου να βοηθηθεί η διαχείριση της πλατφόρμας.

Στην Εικόνα 3.4 παρουσιάζεται η ιεραρχία των κλάσεων του Worker API. Για απλότητα μόνο ένα μέρος των κλάσεων παρουσιάζεται.



Εικόνα 3.4: Η ιεραρχία των κλάσεων των Workers

Οι σημαντικότερες κλάσεις και διεπαφές για την επικοινωνία είναι οι ακόλουθες:



Διεπαφή IWorker: Ορίζει τις βασικές λειτουργίες που πρέπει να υλοποιεί ένας worker.

Κλάση Worker: Η abstract κλάση Worker προσφέρει βοηθητικές μεθόδους που χρησιμοποιούνται σχεδόν από όλους τους Workers όπως ανταλλαγή μηνυμάτων και έλεγχο για το αν έχει ολοκληρωθεί η επικοινωνία.

Exception ResultNotReadyException: Η εξαίρεση αυτή εμφανίζεται όταν κάποιο συστατικό της πλατφόρμας προσπαθεί να πάρει το αποτέλεσμα της επικοινωνίας από έναν Worker ενώ η διαδικασία της επικοινωνίας δεν έχει ολοκληρωθεί.

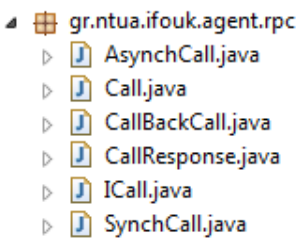
Μηχανισμοί επικοινωνίας ανάμεσα στους πράκτορες

Η επικοινωνία μεταξύ των πρακτόρων η ανταλλαγή SOAP μηνυμάτων ακολουθεί τρία διαφορετικά μοντέλα. Το πρώτο αφορά απλές σύγχρονες κλήσεις. Η κλήση σε ένα απομακρυσμένο πράκτορα επιστρέφει όταν το αποτέλεσμα είναι έτοιμο. Το κύριο πλεονέκτημα αυτού του μηχανισμού είναι ότι μετά την κλήση το αποτέλεσμα είναι πάντα έτοιμο αλλά ο πράκτορας μπλοκάρεται μέχρι να γίνει αυτό. Η δεύτερη μέθοδος είναι η

ασύγχρονη. Στην περίπτωση αυτή ο πράκτορας δημιουργεί μια κλήση με τρόπο όμοιο με τη σύγχρονη κλήση, με τη διαφορά ότι η εκτέλεση του πράκτορα μπλοκάρεται σε αυτή την περίπτωση. Έπειτα ο πράκτορας μπορεί να τσεκάρει περιοδικά αν το αποτέλεσμα της κλήσης είναι έτοιμο. Τέλος η πλατφόρμα παρέχει τον τύπο κλήσης call-back-call όπου ο πράκτορας όχι μόνο μορφοποιεί την κλήση αλλά δηλώνει και τη μέθοδο που θα καλεστεί όταν ανακτήσει το αποτέλεσμα. Ο εκτέλεση του πράκτορα δε μπλοκάρεται και σε αυτή την περίπτωση.

Από την πλευρά του προγραμματιστή ο πράκτορας απλά στιγμιοτυποποιεί κάποια από τις κλάσεις που επεκτείνουν τη κλάση Call (βλέπε Εικόνα 3.5) και την περνάει ως όρισμα στη μέθοδο makeCall() της κλάσης ενθυλάκωσης (wrapper) του πράκτορα. Θα πρέπει να σημειωθεί ότι σε αυτόνομες εφαρμογές όταν ένα αντικείμενο περνιέται ως παράμετρος, η παράμετρος συμπεριφέρεται σαν να είχε περαστεί ως αναφορά (στην πραγματικότητα περνιέται ένα αντίγραφο της παραμέτρου). Αντίθετα στο SOAP, όλες οι παράμετροι αντιγράφονται και περνιούνται με τη SOAP κλήση στον εξυπηρετητή και κάθε τροποποίηση που γίνεται σε αυτές δεν επιστρέφεται ποτέ στον πελάτη. Αυτή η συμπεριφορά είναι ίδια όταν γίνεται χρήση του RMI API (Remote Method Invocation) της Java.

Οι παραπάνω λειτουργίες υλοποιήθηκαν σε κώδικα με τις ακόλουθες κλάσεις και διεπαφές:



Διεπαφή ICall: Ορίζει τις βασικές λειτουργίες που πρέπει να έχει ένα αντικείμενο το οποίο αναπαριστά κλήση σε έναν άλλο πράκτορα (για παράδειγμα ορισμό του είδους τους μηνύματος και των παραμέτρων που αυτό θα περιέχει).

Κλάση Call: Η abstract κλάση Call υλοποιεί τις βασικές λειτουργίες που ορίζονται στη διεπαφή ICall αλλά δεν μπορούν να υπάρξουν στιγμιότυπά της μιας και δεν ορίζει συγκεκριμένο μοντέλο επικοινωνίας. Αυτό πραγματοποιείται στις υποκλάσεις της.

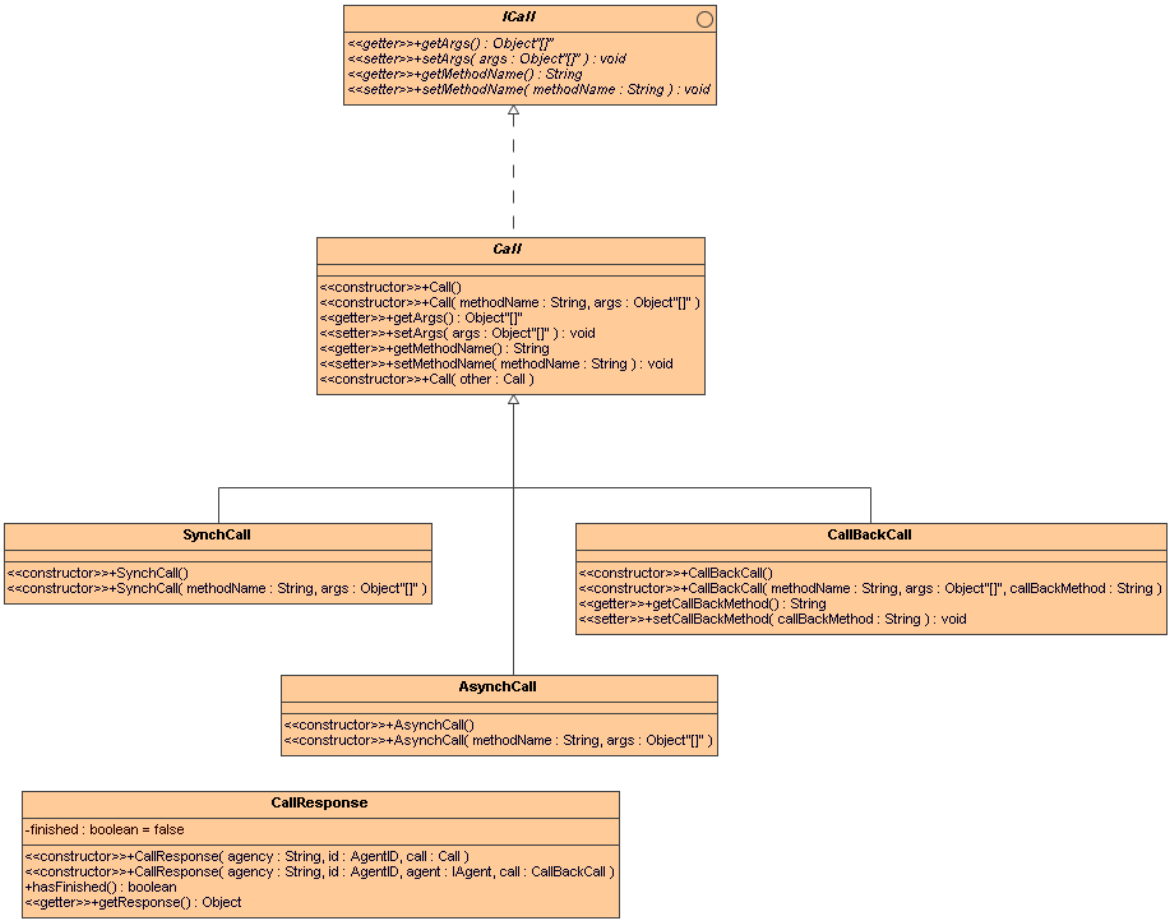
Κλάση SynchCall: Υποκλάση της Call που υλοποιεί τις απαραίτητες λειτουργίες για σύγχρονη επικοινωνία ανάμεσα σε δύο πράκτορες.

Κλάση ASynchCall: Υποκλάση της Call που υλοποιεί τις απαραίτητες λειτουργίες για ασύγχρονη επικοινωνία ανάμεσα σε δύο πράκτορες.

Κλάση CallBackCall: Υποκλάση της Call που υλοποιεί τις απαραίτητες λειτουργίες για επικοινωνία ανάμεσα σε δύο πράκτορες χρησιμοποιώντας .

Κλάση CallResponse: Ενθυλακώνει το αποτέλεσμα μιας κλήσης. Σε περίπτωση που η κλήση ακολουθεί το σύγχρονο μοντέλο επικοινωνίας, τότε το αποτέλεσμα είναι έτοιμο με

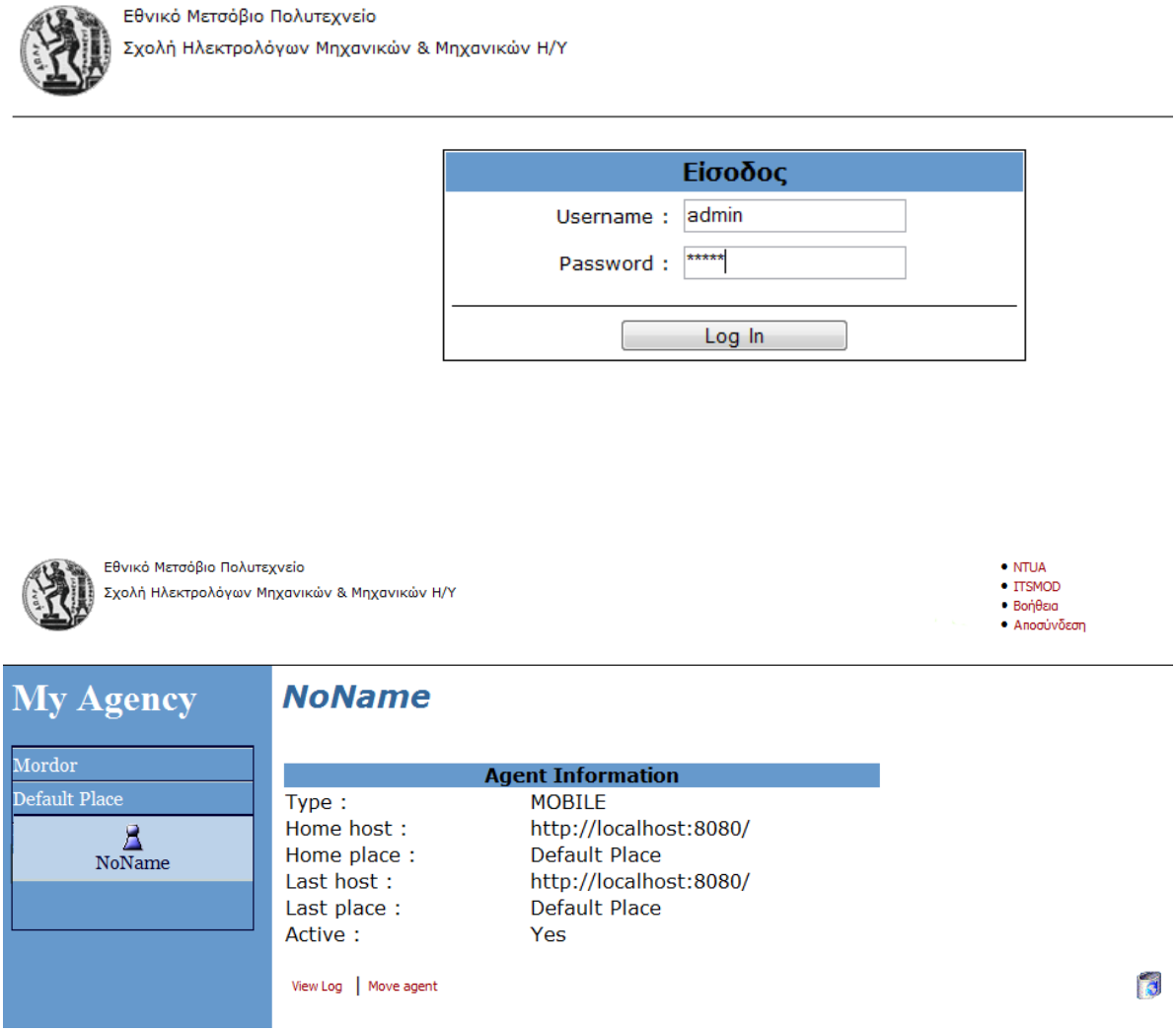
το που ολοκληρώνεται η κλήση. Στην περίπτωση της ασύγχρονης κλήσης και του μοντέλου call-back-call, ο πράκτορας μπορεί να ελέγχει περιοδικά αν το αποτέλεσμα είναι έτοιμο.



Εικόνα 3.5: Διάγραμμα κλάσεων του πακέτου RPC

3.3 Η ΠΛΕΥΡΑ ΤΟΥ ΠΕΛΑΤΗ

Η πλευρά πελάτη της πλατφόρμας SOAP ελέγχεται από διάφορες σελίδες JSP (Java Server Pages), οι οποίες επιτρέπουν την απομακρυσμένη διαχείριση των περιοχών, των πρακτορείων και των πρακτόρων. Οι διαχειριστικές υπηρεσίες επιτρέπουν την παρακολούθηση και τον έλεγχο των πρακτόρων και τις θέσεις (places) ενός πρακτορείου από τους χρήστες. Είναι πιθανό, μεταξύ των άλλων, να δημιουργηθούν, να διαγραφούν και να ανασταλούν πράκτορες, υπηρεσίες, και θέσεις, να ανακτηθούν πληροφορίες για συγκεκριμένους πράκτορες και υπηρεσίες, να εμφανιστεί λίστα με όλους τους πράκτορες που κατοικούν σε μια συγκεκριμένη θέση κ.τ.λ. Πάλι, ο ενσωματωμένος κώδικας σε αυτές τις σελίδες έχει πρόσβαση στις μεθόδους της περιοχής και των πρακτορείων μέσω του API των Workers.



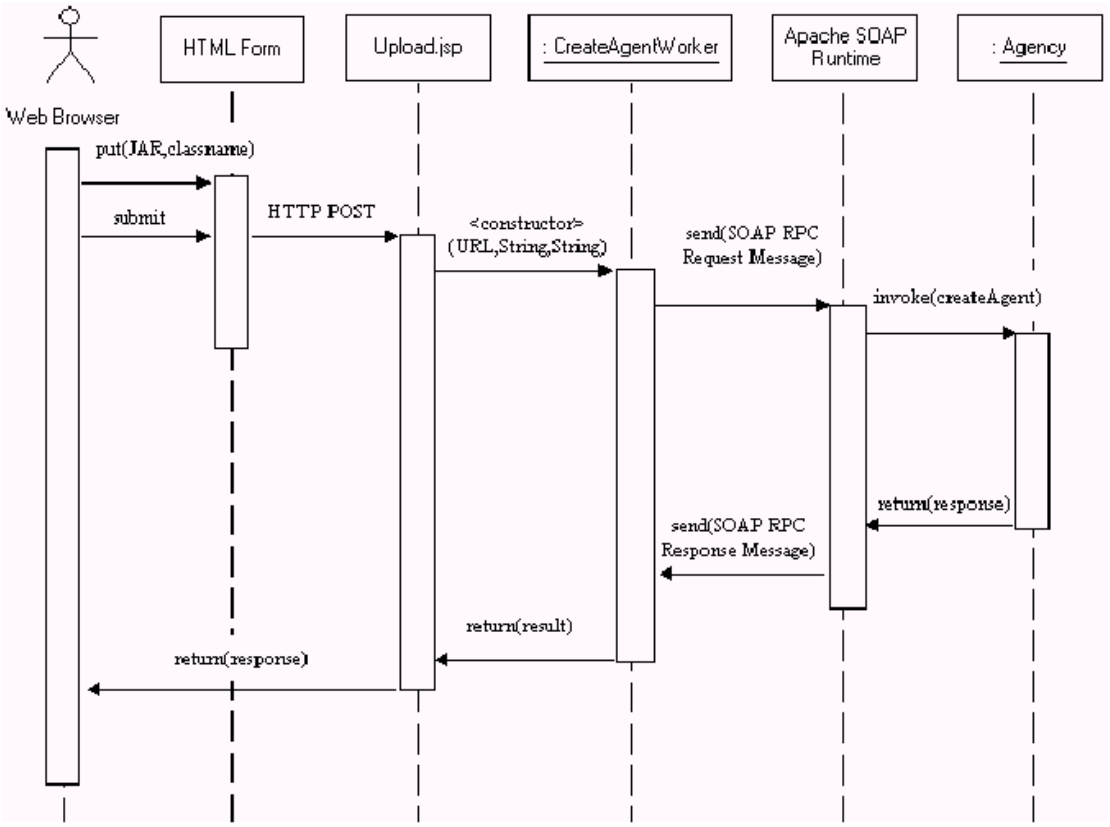
Εικόνα 3.6: Παραδείγματα σελίδων JSP για τη διαχείριση της πλατφόρμας

Όταν ο χρήστης επιλέξει το κουμπί “Submit’, μια σελίδα τύπου JSP ενεργοποιείται στην πλευρά του εξυπηρετητή, η οποία είναι υπεύθυνη για τη προσπέλαση της υπηρεσίας "περιοχής", να συγκεντρώσει όλη την πληροφορία σχετικά με τα καταχωρημένα αντικείμενα σε αυτήν και να τα παρουσιάσει στο χρήστη. Για να αποκτήσει πρόσβαση στις υπηρεσίες της περιοχής, αυτή η JSP σελίδα δημιουργεί ένα αντικείμενο της κλάσης RegionStructureWorker με αναφορά στο URL του rpcrouter servlet. Έπειτα, η JSP σελίδα καλεί τη μέθοδο getResult() για αυτό το αντικείμενο. Όταν κληθεί αυτή η μέθοδος, ο worker περνά της SOAP αίτηση στο rpcrouter servlet το οποίο καλεί την μέθοδο της getStructure() κλάσης περιοχής. Το rpcrouter επιστρέφει το αποτέλεσμα της μεθόδου ως μήνυμα SOAP στο αντικείμενο RegionStructureWorker το οποίο με τη σειρά του το μετασχηματίζει στον κατάλληλο τύπο αντικειμένου. Το αποτέλεσμα είναι πλέον διαθέσιμο

στην JSP σελίδα ώστε να παρουσιάσει στο χρήστη την πληροφορία της περιοχής. Παρόμοιες διαδικασίες ακολουθούνται όταν καλούνται και οι υπόλοιπες πληροφορίες των πρακτορείων και περιοχών.

3.3.1 ΦΟΡΤΩΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ ΤΟΥ ΠΡΑΚΤΟΡΑ

Ο προγραμματιστής υλοποιεί μια εφαρμογή κινητών πρακτόρων βασιζόμενος στο σετ των κλάσεων του API της πλατφόρμας. Μία από τις κλάσεις αυτές θα πρέπει οπωσδήποτε να επεκτείνει την κλάση StationaryAgent ή MobileAgent και το σύνολο των κλάσεων θα πρέπει να συμπιεστεί στη μορφή JAR της Java. Με χρήση μιας HTML σελίδας, το αρχείο και το όνομα της βασικής κλάσης του πράκτορα αποστέλλονται σε μια σελίδα JSP. Η JSP σελίδα από τη μεριά της στιγμιοτυποποιεί ένα αντικείμενο της κλάσης CreateAgentWorker η οποία κάνει μια SOAP RPC αίτηση στο πρακτορείο για τη δημιουργία του πράκτορα. Η διαδικασία αυτή φαίνεται στην Εικόνα 3.7.



Εικόνα 3.7 Διάγραμμα κατάστασης κατά το φόρτωμα ενός πράκτορα

Έπειτα το πρακτορείο φορτώνει την κλάση του πράκτορα και δημιουργεί ένα ενθυλακωτή γι' αυτόν. Τέλος δημιουργείται ένα νέο ThreadGroup το οποίο αναλαμβάνει την εκτέλεση του πράκτορα.

3.3.2 ΚΙΝΗΤΙΚΟΤΗΤΑ

Οι πράκτορες σε αυτή την πλατφόρμα διαχωρίζονται σε δύο κύριες κατηγορίες: στους κινητούς και στους στατικούς. Ενώ οι τελευταίοι παραμένουν στο πρακτορείο που τους δημιούργησε μέχρι το τέλος της εκτέλεσής τους, οι πρώτοι έχουν την ικανότητα να μεταναστεύουν σε απομακρυσμένα πρακτορεία αυτόνομα ή να τους δοθεί εντολή γι' αυτό. Αυτή η μεταφορά επιτυγχάνεται και από τα δύο πρακτορεία με διαπραγμάτευση και εάν είναι δυνατή η μεταφορά τότε την πραγματοποιούν. Κάθε πράκτορας (όπως και κάθε πρόγραμμα) αποτελείται από δύο μέρη: την πραγματική εντολή στην μηχανή (π.χ. τις εντολές assembly ή τους Java bytecodes), και το τρέχον αποτύπωμα στη μνήμη. Και τα δύο αυτά μέρη μεταφέρονται στο απομακρυσμένο πρακτορείο ώστε να καταστεί δυνατή η επαναδημιουργία του πράκτορα.

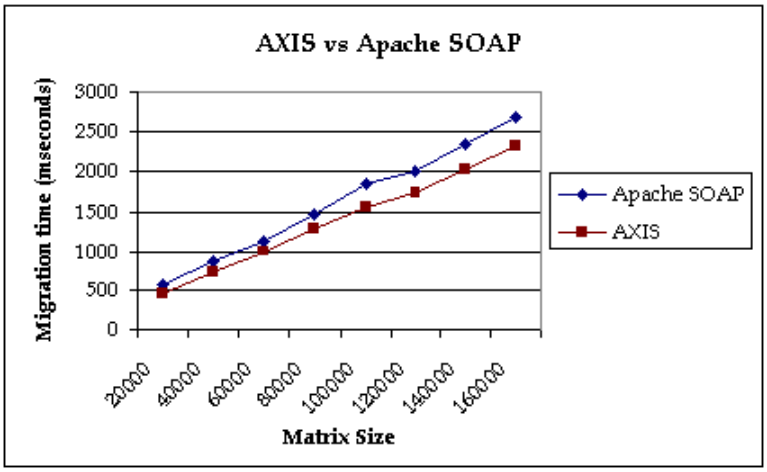
3.4 ΕΠΕΚΤΑΣΗ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ ΚΙΝΗΤΩΝ ΠΡΑΚΤΟΡΩΝ

3.4.1 ΜΕΤΑΒΑΣΗ ΣΤΗ ΒΙΒΛΙΟΘΗΚΗ AXIS

Η πρώτη έκδοση της πλατφόρμας κινητών πρακτόρων χρησιμοποιούσε τη βιβλιοθήκη Apache SOAP για την ανταλλαγή μηνυμάτων SOAP ανάμεσα στα συστατικά της. Το επόμενο βήμα στην ανάπτυξη της πλατφόρμας ήταν η αξιοποίηση των δυνατοτήτων της βιβλιοθήκης AXIS (Apache eXtensible Interaction System). Στην πράξη, το μόνο κομμάτι της πλατφόρμας που χρειάστηκε να πραγματοποιηθεί κάποια αλλαγή ήταν το Workers API, μιας και οι Workers κρύβουν την ανταλλαγή των μηνυμάτων από τα υπόλοιπα συστατικά. Το AXIS όχι μόνο έχει εσωτερικά μια εντελώς διαφορετική αρχιτεκτονική σε σχέση με το Apache SOAP, αλλά επιπλέον χρησιμοποιεί τη προδιαγραφή SAX για να επεξεργάζεται μηνύματα σε μορφή XML, σε αντίθεση με το Apache SOAP που χρησιμοποιεί την προδιαγραφή DOM. Η κύρια διαφορά ανάμεσα στις δύο προδιαγραφές είναι ότι το SAX επεξεργάζεται τα κείμενα XML σειριακά, χωρίς να χρειάζεται να αποθηκεύσει όλα τα δεδομένα στη μνήμη, ενώ το DOM αναπαριστά όλο το κείμενο σε μορφή δέντρου. Αυτή η διαφορά έχει σημαντικό αντίκτυπο στην επιδοση της πλατφόρμας.

Προκειμένου να μετρηθεί η διαφορά της απόδοσης ανάμεσα στις δύο πλατφόρμες, χρησιμοποιήθηκε η ακόλουθη τεχνική. Αναπτύχθηκε ένας κινητός πράκτορας, ο οποίος

περιέχει ένα «φορτίο». Το φορτίο αυτό είναι ένας πίνακας από bytes, μεταβλητού μεγέθους. Ο κινητός πράκτορας ξεκινάει την εκτέλεσή του από ένα πρακτορείο, μεταβαίνει σε κάποιο άλλο και επιστρέφει στο αρχικό πρακτορείο. Η διαδικασία αυτή επαναλαμβάνεται για μεταβλητά φορτία, και για έναν μεγάλο αριθμό επαναλήψεων ανά φορτίο. Σαν μέτρο επίδοσης χρησιμοποιούμε τη μέση τιμή του συνολικού χρόνου της διαδικασίας μετανάστευσης.



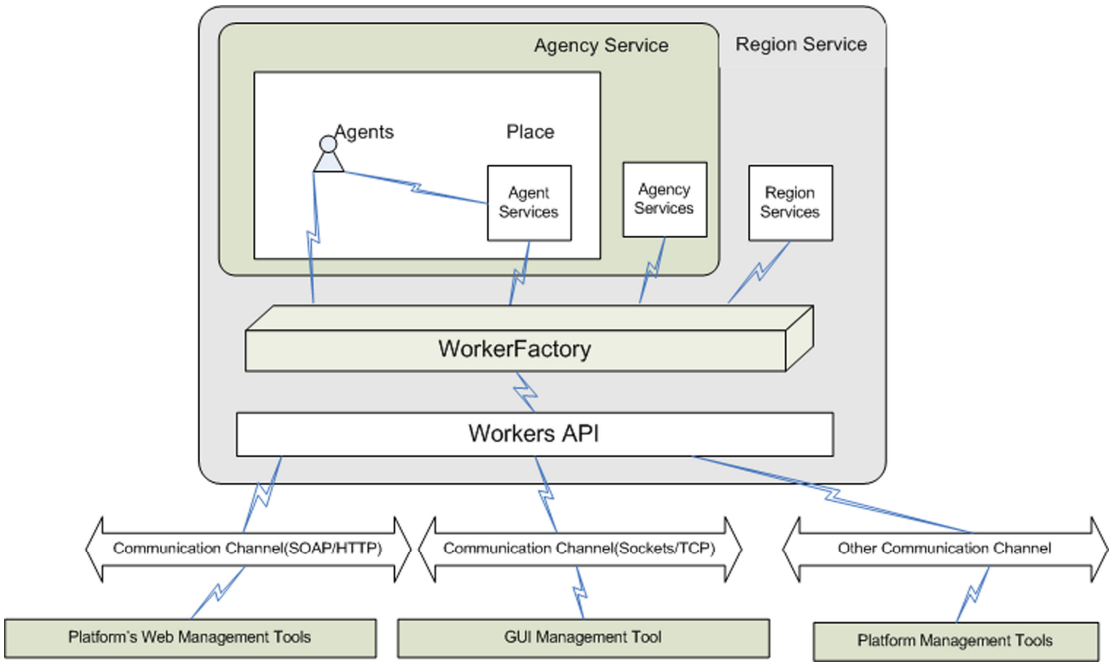
Εικόνα 3.8: Σύγκριση Apache SOAP και AXIS

Όπως φαίνεται στην Εικόνα 3.8, σε όλες τις περιπτώσεις η απόδοση της πλατφόρμας είναι καλύτερη όταν χρησιμοποιείται η βιβλιοθήκη AXIS. Η διαφορά αυξάνεται όσο αυξάνει και το μέγεθος του φορτίου (μέγεθος αποτυπώματος του πράκτορα στη μνήμη). Το αποτέλεσμα αυτό ήταν αναμενόμενο, μιας και η βιβλιοθήκη AXIS χρησιμοποιεί τη προδιαγραφή SAX, η οποία είναι και πιο γρήγορη.

3.4.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΑΝΕΞΑΡΤΗΤΗ ΑΠΟ ΤΟ ΣΤΡΩΜΑ ΜΕΤΑΦΟΡΑΣ ΔΕΔΟΜΕΝΩΝ

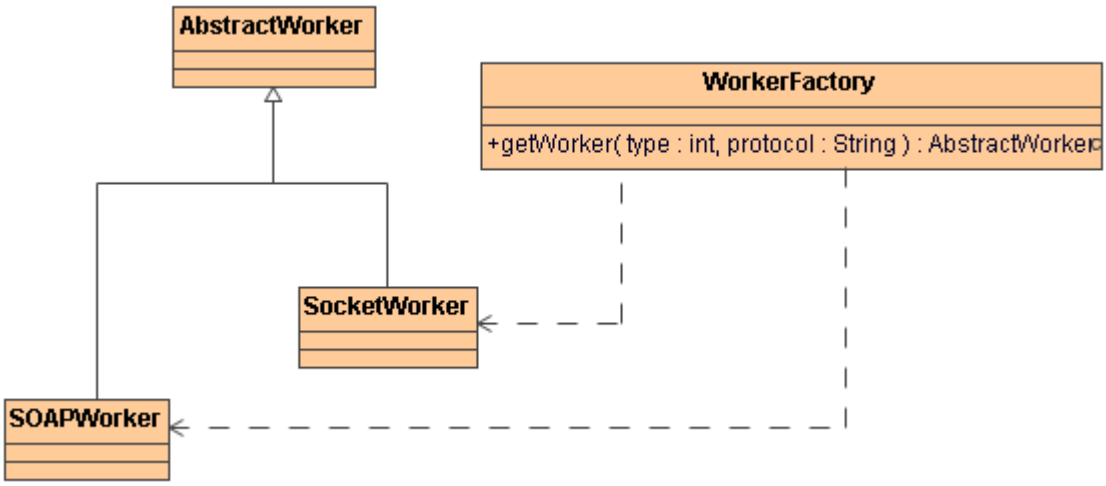
Η αρχιτεκτονική της πλατφόρμας των κινητών πρακτόρων είχε το μειονέκτημα ότι παρόλο που η αντικατάσταση του στρώματος μεταφοράς δεδομένων ήταν εύκολη, μπορούσε να χρησιμοποιηθεί μόνο ένας τύπος καναλιού επικοινωνίας κάθε φορά. Το γεγονός αυτό περιόριζε την διαλειτουργικότητα ανάμεσα σε εκδόσεις της πλατφόρμας που χρησιμοποιούνταν σε διαφορετικό περιβάλλον εκτέλεσης.

Η λύση που προτιμήθηκε είναι η αντικατάσταση του στρώματος μεταφοράς με ένα πιο ευέλικτο, το οποίο επιτρέπει την ταυτόχρονη χρησιμοποίηση παραπάνω από ενός καναλιού μεταφοράς δεδομένων. Η παραλλαγή της αρχικής αρχιτεκτονικής της πλατφόρμας φαίνεται στην Εικόνα 3.9.



Εικόνα 3.9: Αρχιτεκτονική ανεξάρτητη από το στρώμα μεταφοράς δεδομένων

Ο τρόπος που η κάθε οντότητα της πλατφόρμας επικοινωνεί με τα υπόλοιπα συστατικά της παραμένει ο ίδιος. Χρησιμοποιώντας τον κατάλληλο Worker γίνεται αποστολή του μηνύματος. Αυτό που διαφοροποιείται είναι ο τρόπος με τον οποίο η οντότητα αποκτά πρόσβαση στον κατάλληλο πράκτορα. Η νέα μέθοδος χρησιμοποιεί το σχεδιαστικό πρότυπο των «εργοστασίων» (factory design pattern).



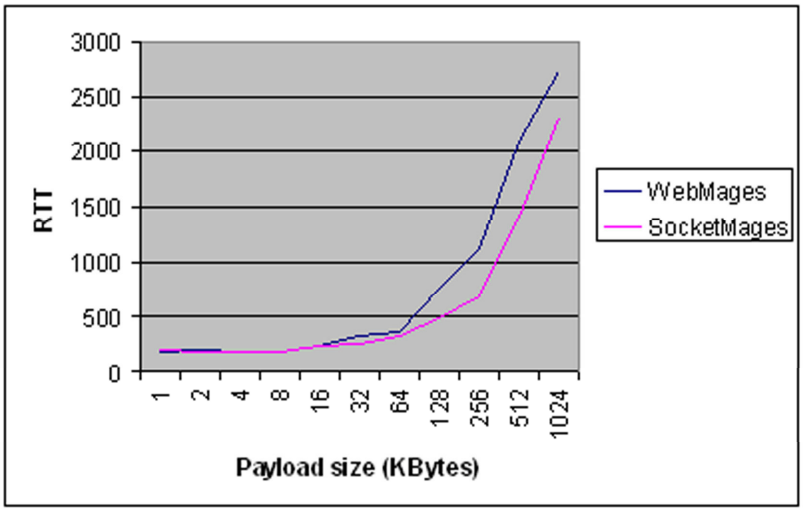
Εικόνα 3.10: Σχεδιαστικό πρότυπο των «εργοστασίων»

Σύμφωνα με το σχεδιαστικό πρότυπο των «εργοστασίων», όλα τα αντικείμενα που υλοποιούν ένα συγκεκριμένο interface ή abstract κλάση, και κατασκευάζονται από ένα συγκεκριμένο αντικείμενο, το οποίο ονομάζεται «αντικείμενο-εργοστάσιο». Στην

περίπτωσή μας, η abstract κλάση είναι η AbstractWorker, η οποία ορίζει τη βασική λειτουργικότητα των Workers. Το αντικείμενο-εργοστάσιο υλοποιείται από την κλάση WorkerFactory. Όταν κάποιο αντικείμενο θέλει να αποκτήσει πρόσβαση σε κάποιον Worker, τότε καλεί τη μέθοδο getWorker() της κλάσης WorkerFactory. Οι παράμετροι αυτής της μεθόδου είναι η διεύθυνση αποστολής του μηνύματος, τα δεδομένα που θα μεταφερθούν με το μήνυμα και ο τύπος των δεδομένων. Βάσει αυτών των πληροφοριών, η μέθοδος επιστρέφει τον κατάλληλο Worker.

Προκειμένου να δοκιμαστεί η παραπάνω αρχιτεκτονική, υλοποιήθηκε ένα καινούργιο σύνολο από Workers, οι οποίοι χρησιμοποιούν TCP/IP sockets για την επικοινωνία. Οι λόγοι για αυτή την επιλογή είναι πολλοί. Πρώτα απ' όλα, η χρησιμοποίηση των sockets αναμένεται να οδηγήσει σε βελτίωση της απόδοσης, μιας και δεν απαιτείται η κωδικοποίηση των δεδομένων σε Base64 μορφή για τη μεταφορά τους. Επιπλέον, το μήνυμα που μεταφέρεται είναι μόνο τα απαραίτητα δεδομένα και όχι π.χ. ολόκληρο XML κείμενο.

Προκειμένου να δοκιμαστούν τα παραπάνω, χρησιμοποιήσαμε την ίδια λογική αξιολόγησης της πλατφόρμας όπως και με την σύγκριση Apache SOAP – AXIS. Τα αποτελέσματα της σύγκρισης Apache SOAP – sockets φαίνονται στην Εικόνα 3.11 Για μικρά φορτία (1-64 Kbytes), ο μέσος χρόνος είναι περίπου ίδιος. Ωστόσο όσο αυξάνεται το μέγεθος του φορτίου, τόσο αυξάνεται και η διαφορά ανάμεσα στις δύο υλοποιήσεις, με την υλοποίηση των sockets να είναι πιο γρήγορη.

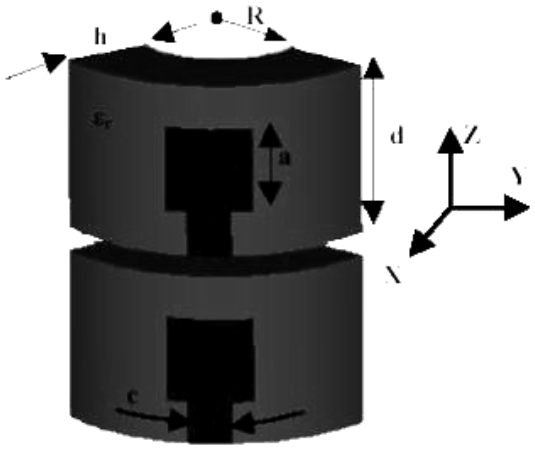


Εικόνα 3.11: Σύγκριση Apache SOAP και sockets

3.5 ΕΦΑΡΜΟΓΕΣ

3.5.1 ΠΡΟΣΟΜΟΙΩΣΗ ΣΥΜΜΟΡΦΩΝ ΣΤΟΙΧΕΙΟΚΕΡΑΙΩΝ

Η πλατφόρμα κινητών πρακτόρων έχει χρησιμοποιηθεί σε διάφορες εφαρμογές. Ωστόσο ο κύριος λόγος ανάπτυξής της ήταν για τη χρησιμοποίησή της σε ένα σύστημα κατανεμημένης επίλυσης παραμετρικών προβλημάτων. Ειδικότερα, χρησιμοποιήθηκε για την κατανεμημένη εκτέλεση ενός κώδικα προσομοίωσης σύμμορφων στοιχειοκεραιών.



Εικόνα 3.12: Η γεωμετρία της σύμμορφης στοιχειοκεραίας

Ο κώδικας της προσομοίωσης υπήρχε έτοιμος σε εκτελέσιμη μορφή, και πραγματοποιεί προσομοίωση της κεραίας για συγκεκριμένες τιμές των διαστάσεων της γεωμετρίας που φαίνεται στην Εικόνα 3.12. Το μοντέλο που χρησιμοποιήθηκε για την κατανομή των προσομοιώσεων ήταν αυτό του Αφέντη – Εργάτη. Ένας πράκτορας, ο οποίος ονομάζεται πράκτορας-αφέντης, είναι υπεύθυνος για το συντονισμό των προσομοιώσεων. Ο πράκτορας αυτός τεμαχίζει το συνολικό φόρτο εργασίας, χωρίζοντάς τον σε πολλά σύνολα παραμέτρων. Ύστερα, δημιουργεί ένα σύνολο από πράκτορες-εργάτες, σε κάθε έναν από τους οποίους ανατίθεται ένα συγκεκριμένο σύνολο παραμέτρων. Οι πράκτορες αυτοί μεταβαίνουν σε όλους τους διαθέσιμους απομακρυσμένους κόμβους, όπου εκτελούν τον κώδικα της προσομοίωσης για το σετ των παραμέτρων που τους έχει δοθεί. Όταν ολοκληρωθεί η διαδικασία αυτή, επιστρέφουν τα αποτελέσματα στον πράκτορα-αφέντη. Σε περίπτωση που υπάρχουν διαθέσιμα σύνολα παραμέτρων, ο πράκτορας-αφέντης το στέλνει στον πράκτορα-εργάτη, ο οποίος επαναλαμβάνει το παραπάνω βήμα, αλλιώς ολοκληρώνει την εκτέλεσή του. Όταν ολοκληρωθεί η εκτέλεση του κώδικα για όλα τα σύνολα παραμέτρων, ο πράκτορας-αφέντης συγκεντρώνει όλα τα αποτελέσματα και παράγει το τελικό αποτέλεσμα.

Μία σημαντική παράμετρος της εφαρμογής είναι ότι ο χρήστης της δεν γνωρίζει ότι για την παραγωγή των αποτελεσμάτων της προσομοίωσης χρησιμοποιείται η παραπάνω πλατφόρμα. Η είσοδος των δεδομένων για την εκτέλεση της προσομοίωσης γίνεται μέσω ενός συνόλου ιστοσελίδων. Στη συνέχεια, ένα Java Servlet επεξεργάζεται τα δεδομένα και ξεκινά τη διαδικασία της προσομοίωσης με τους κινητούς πράκτορες.

3.5.2 ΣΥΣΤΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ ΨΗΦΟΦΟΡΙΑΣ

Μια δεύτερη εφαρμογή η οποία αναπτύχθηκε βασισμένη στην πλατφόρμα κινητών πρακτόρων ήταν η υλοποίηση συστήματος ηλεκτρονικής ψηφοφορίας. Σκοπός της εφαρμογής ήταν η ενσωμάτωση μηχανισμών ασφαλείας που επιτρέπουν τη μεταφορά μηνυμάτων μεταξύ δύο πρακτόρων, προκειμένου να εξασφαλίζονται χαρακτηριστικά όπως η ασφάλεια και το απόρρητο της ψήφου.

Το σύστημα ηλεκτρονικής ψηφοφορίας, έπρεπε να ικανοποιεί όσο το δυνατόν περισσότερες από τις απαιτήσεις των εκλογικών συστημάτων που εφαρμόζονται σήμερα, όπως:

- Ατομική εγγραφή και εξουσιοδότηση.
- Προκαθορισμένη διάρκεια της διαδικασίας ψηφοφορίας.
- Διενέργεια των εκλογών στα ειδικά διαμορφωμένα εκλογικά κέντρα.
- Διανομή ψηφοδελτίων με προκαθορισμένους υποψήφιους.
- Επιλογή της εκάστοτε προτίμησης (συνήθως κρυφά).
- Ασφαλής συγκέντρωση ψηφοδελτίων για αμερόληπτη καταμέτρηση.

Αρχική προϋπόθεση είναι η ψηφοφορία να γίνεται μέσα στο έγκυρο χρονικό διάστημα. Διαφορετικά, όταν ο ψηφοφόρος ζητήσει από το τερματικό την αρχική σελίδα της εφαρμογής ενημερώνεται από το σύστημα ότι δεν μπορεί να ψηφίσει, επειδή έχει προσέλθει είτε πριν την έναρξη είτε μετά το πέρας της ψηφοφορίας.

Κάθε ψηφοφόρος, προκειμένου να έχει δικαίωμα ψήφου, έχει μαζί του μια «έξυπνη κάρτα» (smart card) που περιέχει το πιστοποιητικό που του δίνει την απαραίτητη εξουσιοδότηση. Θεωρούμε ότι έχει εκδοθεί από μια έμπιστη εκδούσα αρχή ειδικά γι αυτό το σκοπό. Κάθε τέτοιο πιστοποιητικό έχει ένα μοναδικό σειριακό αριθμό (serial number) με βάση τον οποίο γίνεται η αντιστοίχιση με το συγκεκριμένο ψηφοφόρο και η καταχώρησή του στο σύστημα. Σε περίπτωση μη ύπαρξης πιστοποιητικού, η διαδικασία δεν προχωράει και με τον τρόπο αυτό, διασφαλίζεται η εξουσιοδοτημένη υποβολή ψήφου.

Σαν πρόσθετη δικλείδα ασφαλείας, υπάρχει αντιστοίχιση κάθε πιστοποιητικού με συγκεκριμένο όνομα χρήστη και κωδικό (username, password). Έτσι, για παράδειγμα, κάποιος που έχει στην κατοχή του «παράνομα» ένα έγκυρο πιστοποιητικό, δεν θα μπορεί να

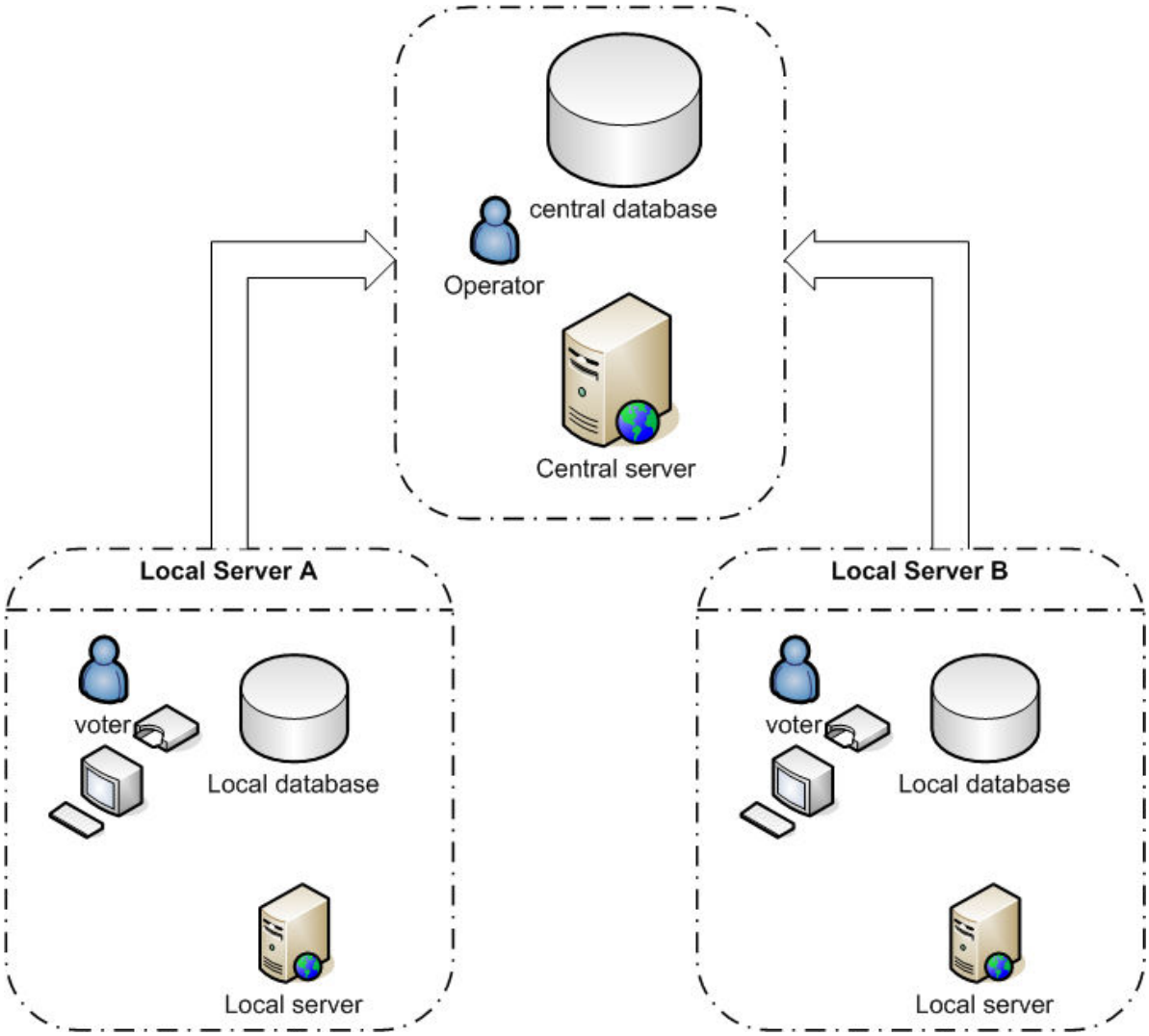
το χρησιμοποιήσει. Τα στοιχεία αυτά βρίσκονται καταχωρημένα τόσο στις βάσεις δεδομένων των εκλογικών κέντρων, όσο και στην κεντρική βάση δεδομένων. Κάθε φορά που κάποιος επιχειρεί να ψηφίσει, γίνεται έλεγχος των παραπάνω στοιχείων στην τοπική βάση δεδομένων και, εφόσον υπάρχει ταίριασμα, καταχώρηση του γεγονότος ότι έχει ψηφίσει, ώστε να μην έχει τη δυνατότητα να ξαναψηφίσει. Κάθε τοπική βάση δεδομένων ενημερώνει ανά τακτά χρονικά διαστήματα την κεντρική, η οποία με τη σειρά τις ενημερώνει και τις υπόλοιπες περιφερειακές βάσεις. Έτσι διασφαλίζεται ότι κάποιος που έχει ήδη ψηφίσει σε ένα εκλογικό κέντρο, δεν θα έχει τη δυνατότητα να ξαναψηφίσει και σε κάποιο άλλο εκλογικό κέντρο. Θεωρούμε ότι τα διαστήματα αυτά είναι επαρκώς μικρά, ώστε να μην προλαβαίνει κανείς να μεταβεί από ένα εκλογικό κέντρο σε κάποιο άλλο. Τέλος, σε περίπτωση μη εισαγωγής των παραπάνω διαπιστευτηρίων, η διαδικασία «κολλάει», μέχρι αυτά να εισαχθούν.

Σε ό,τι αφορά τη διαδικασία μεταφοράς του μηνύματος, αυτό θα καταρχάς κρυπτογραφείται με τον αλγόριθμο κρυπτογράφησης RSA, ώστε να αποτραπεί η ανάγνωσή του κατά τη μεταφορά. Αφού κρυπτογραφηθεί, υπογράφεται από τον τοπικό server και όταν το μήνυμα φτάσει στον κεντρικό server, πριν την αποκρυπτογράφησή του, γίνεται επαλήθευση της υπογραφής. Έτσι, διαπιστώνεται η ακεραιότητα του ή μη του μηνύματος, καθώς και το αν έχει αποσταλεί πράγματι από server του συστήματος ηλεκτρονικής ψηφοφορίας.

Αξίζει να σημειωθεί ότι το μήνυμα μεταφέρει μόνο την ψήφο, και κανένα άλλο στοιχείο που να σχετίζεται με την ταυτότητα του ψηφοφόρου, ώστε να μην είναι δυνατή η αντιστοίχηση σε κανένα στάδιο της διαδικασίας και να διασφαλίζεται το απόρρητο της ψήφου.

Απαραίτητη παραδοχή για τη συνολική λειτουργία του συστήματος είναι η μη ανάμειξη ανθρώπινου παράγοντα στη διαχείρισή του, τόσο στην κεντρική όσο και στις περιφερειακές μονάδες, ώστε ενδεχόμενη κακόβουλη παρέμβαση να μην αφορά τα σημεία διαχείρισης.

Η βασική αρχιτεκτονική του συστήματος ηλεκτρονικής ψηφοφορίας φαίνεται στην Εικόνα 3.13:



Εικόνα 3.13: Αρχιτεκτονική συστήματος ηλεκτρονικής ψηφοφορίας

Το σύστημα αποτελείται από ένα κεντρικό εξυπηρετητή, αποτελούμενο κατά βάση από ένα εξυπηρετητή ιστού και μία βάση δεδομένων, καθώς και από πολλά περιφερειακά συστήματα, τα οποία αντιστοιχούν στα εκλογικά κέντρα. Ο κεντρικός εξυπηρετητής αναλαμβάνει τη συγκέντρωση των ψήφων, ενώ βρίσκεται σε διαρκή επικοινωνία με τα περιφερειακά. Τα περιφερειακά συστήματα των εκλογικών κέντρων είναι αυτά στα οποία οι ψηφοφόροι, μέσω ενός τερματικού, καταθέτουν την ψήφο τους. Αποτελούνται από έναν τοπικό εξυπηρετητή ιστού και μια τοπική βάση δεδομένων.

Η μεταφορά την ψήφου γίνεται με έναν κινητό πράκτορα, τον CryptoAgent, η κλάση του οποίου αποτελεί επέκταση της κλάσης MobileAgent της πλατφόρμας κινητών αντιπροσώπων. Στη μέθοδο start() του πράκτορα επιτελούνται επιπλέον οι ακόλουθες λειτουργίες:

- Κρυπτογράφηση της ψήφου που θα μεταφερθεί πριν την μεταφορά του πράκτορα
- Αποκρυπτογράφηση της ψήφου μετά τη μεταφορά
- Δημιουργία και έλεγχος της γνησιότητας της ψηφιακής υπογραφής

3.6 ΣΥΜΠΕΡΑΣΜΑΤΑ

Η πλατφόρμα κινητών πρακτόρων που παρουσιάστηκε σε αυτό το κεφάλαιο είχε σαν στόχο να προσφέρει ένα προσαρμόσιμο και επεκτάσιμο περιβάλλον εκτέλεσης για κινητούς πράκτορες. Η επιτυχία της προσέγγισης γίνεται φανερή από τις διαφορετικές εφαρμογές οι οποίες αναπτύχθηκαν πάνω από το μεσισμικό.

Το μοντέλο υπηρεσιών Ιστού ταιριάζει με τις αρχιτεκτονικές απαιτήσεις της πλατφόρμας, η οποία μπορεί να χρησιμοποιηθεί σε ήδη υπάρχοντες εξυπηρετητές ιστού. Με αυτόν τον τρόπο είναι δυνατή η χρησιμοποίηση της υπάρχουσας υποδομής και των πόρων της σε περιπτώσεις που αυτοί παραμένουν ανενεργοί, ενώ επιπλέον ευφυΐα μπορεί να προστεθεί σε αυτούς του κόμβους. Τέλος, η δυνατότητα διαχείρισης της πλατφόρμας μέσω μίας εφαρμογής ιστού διευκολύνει σε μεγάλο βαθμό την απομακρυσμένη διαχείριση του συστήματος και των επιμέρους συστατικών του.

4 ΠΛΕΓΜΑ - GRID

Παρόλο που η τεχνολογία των κινητών πρακτόρων προσφέρει πολλά εργαλεία για την ανάπτυξη κατανεμημένων εφαρμογών αντιμετωπίζει ένα σημαντικό πρόβλημα: την αξιοποίηση ιδιαίτερα μεγάλου αριθμού πόρων. Παρόλο που έχουν προταθεί αλγόριθμοι και εργαλεία που επιτρέπουν τον καλύτερο συντονισμό και διαχείριση των κινητών πρακτόρων και των συστατικών των πλατφόρμων, εμπόδια όπως η έλλειψη μιας συγκεκριμένης γλώσσας επικοινωνίας και η ασυμβατότητα ανάμεσα στις διάφορες πλατφόρμες κινητών πρακτόρων κάνουν απαγορευτική την χρησιμοποίηση των κινητών πρακτόρων για κατανεμημένες εφαρμογές πολύ μεγάλης κλίμακας.

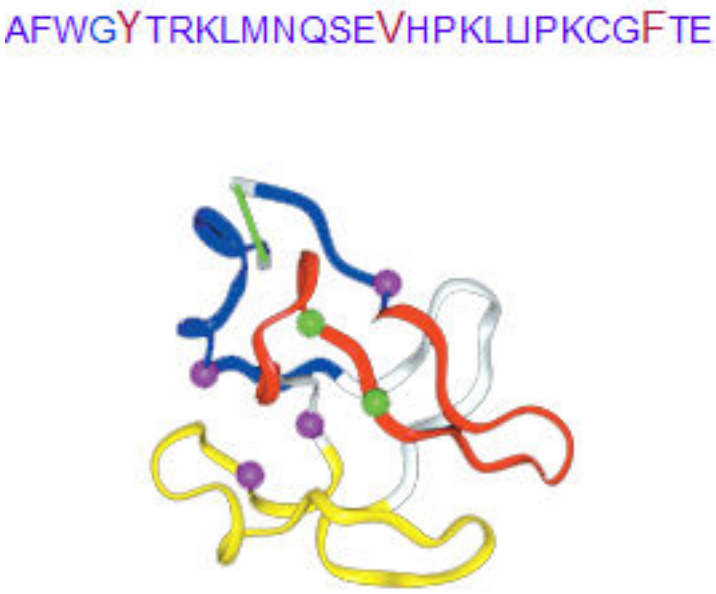
Το συγκεκριμένο κενό έρχεται να καλύψει η τεχνολογία του Πλέγματος. Το Πλέγμα είναι μια τεχνολογία η οποία επιτρέπει σε χρήστες από όλο τον κόσμο να χρησιμοποιήσουν πόρους οι οποίοι βρίσκονται διάσπαρτοι σε ολόκληρο τον κόσμο. Μεγάλο μέρος της πολυπλοκότητας που διέπει το συντονισμό όλων αυτών των κατανεμημένων πόρων κρύβεται με τη χρήση εργαλείων όπως είναι οι δικτυακές πλεγματικές πύλες και διάφορες άλλες γραφικές διεπαφές χρήστη.

Προκειμένου να διερευνηθούν οι δυνατότητες της τεχνολογίας του Πλέγματος στα πλαίσια της ερευνητικής εργασίας αναπτύχθηκαν κατάλληλα εργαλεία για την μεταφορά μιας ήδη υπάρχουσας εφαρμογής στο Πλέγμα. Η εφαρμογή πραγματεύεται την απαρχής πρόβλεψη της αναδίπλωσης των πρωτεϊνών στην τρισδιάστατη δομή τους στο χώρο [70]. Ο αλγόριθμος που έχει αναπτυχθεί στο Εργαστήριο Γενετικής του Γ.Π.Α. και με μόνη πληροφορία την αμινοξική ακολουθία μίας πρωτεΐνης, μπορεί να προβλέψει τα αμινοξέα που έχουν κρίσιμη σημασία για το σχηματισμό του πυρήνα που σταθεροποιεί την τριτοταγή δομή και να υπολογίζει τα αμινοξέα που έχουν την τάση να σχηματίσουν τον πυρήνα της πρωτεϊνικής δομής, τα οποία ονομάζονται Most Interacting Residues (MIR) (Εικόνα 4.1).

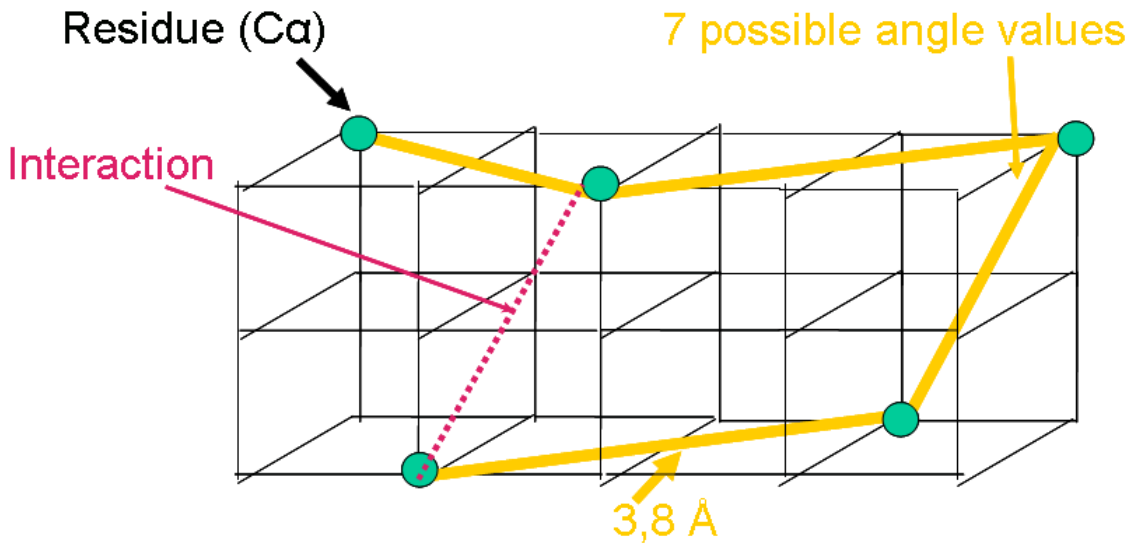
Ο αλγόριθμος αυτός μελετά τα πρώτα στάδια της διαδικασίας του διπλώματος της πρωτεΐνης. Αρχικά η ακολουθία θεωρείται ελεύθερη στο χώρο, με κάθε ένα από τα αμινοξέα να μπορεί να λάβει θέση σε ένα τρισδιάστατο πλέγμα. Χρησιμοποιώντας τεχνικές προσομοίωσης Monte Carlo ελέγχονται διαφορετικές μεταβολές στην αλυσίδα, πραγματοποιώντας μία μόνο αλλαγή κάθε φορά. Τα κριτήρια που χρησιμοποιούνται για την επιλογή της νέας θέσης για το κάθε αμινοξύ είναι τα ακόλουθα:

- Προηγούμενη θέση
- Ενέργεια αλληλεπίδρασης με τα κοντινά αμινοξέα
- Απόσταση από τα αμινοξέα με τα οποία είναι συνδεδεμένο

- Συνολική ενέργεια της νέας δομής

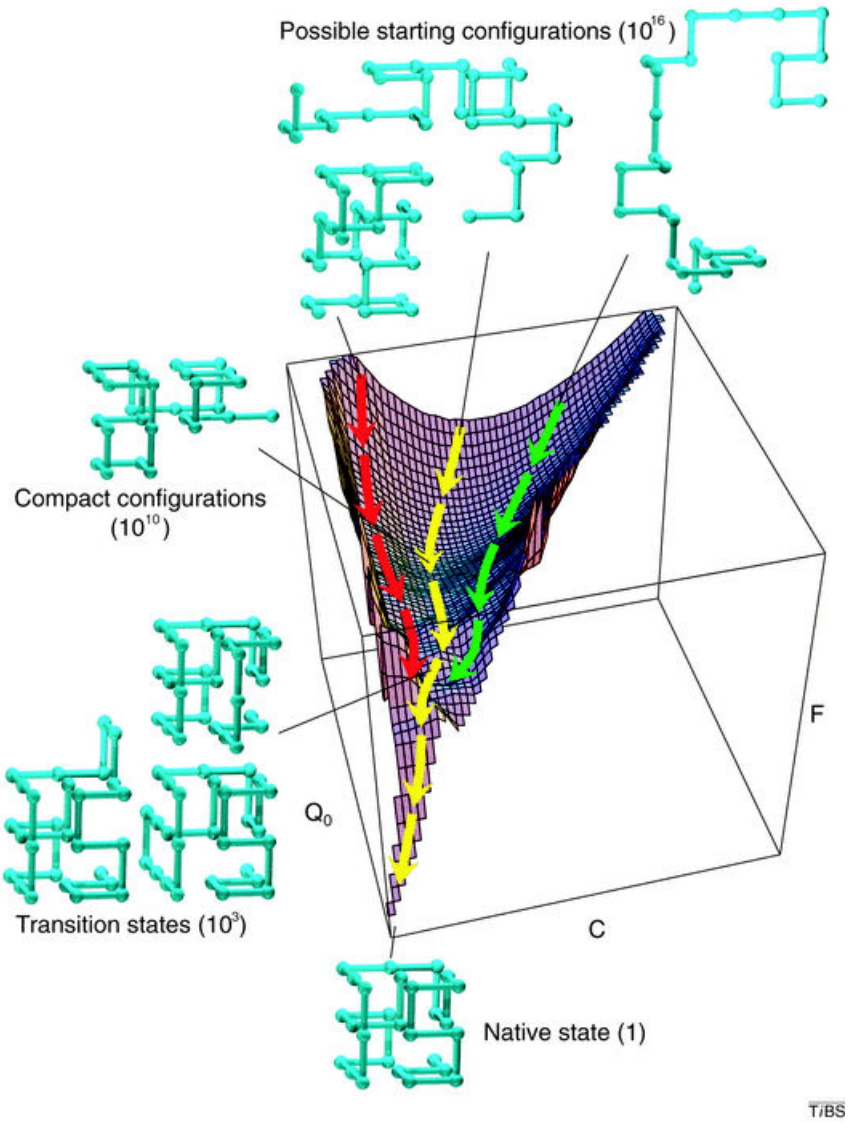


Εικόνα 4.1: Τρισδιάστατη δομή ακολουθίας και MIR.



Εικόνα 4.2: Τρισδιάστατη δομή ακολουθίας σε διακριτές θέσεις.

Ειδικότερος στόχος ήταν η εφαρμογή του αλγορίθμου σε ένα αρκετά μεγάλο πλήθος αντιπροσωπευτικών πρωτεϊνικών ώστε να αξιολογηθεί και βελτιωθεί η προγνωστική του δυνατότητα. Ακολουθίες με μεγάλο μήκος συχνά μπορούν να αναλυθούν ως επί μέρους ακολουθίες με μικρότερο μέγεθος, σαν αρχικά δεδομένα του προβλήματός μας θεωρήθηκαν ακολουθίες με μήκος μικρότερο ή ίσο από 150 αμινοξέα.



Εικόνα 4.3: Δίπλωμα ακολουθίας

Επειδή ο χρόνος υπολογισμού αυξάνει μη γραμμικά με το μέγεθος της πρωτεΐνης, καθιστώντας την εφαρμογή του αλγορίθμου ακόμα πιο απαιτητική για μεγάλα μόρια (4 CPU ώρες για μία πρωτεΐνη μέσου μεγέθους, 150-200 αμινοξέων) απαιτείται η κατανεμημένη χρήση υπολογιστικής ισχύος σε Grid (Computational Grid) Η προτεινόμενη μορφή της πλεγματικής μορφής της εφαρμογής θα είναι προσβάσιμη μέσω μίας Πλεγματικής Πύλης (Grid Portal). Η Πύλη αυτή θα επιτρέπει στους χρήστες να υποβάλλουν εργασίες (jobs) στο Πλέγμα για να εκτελεστούν. Ο χρήστης θα αποκτά πρόσβαση στα αποτελέσματα μέσω της Πύλης μόλις αυτά είναι έτοιμα.

4.1 ΔΙΚΤΥΑΚΗ ΠΛΕΓΜΑΤΙΚΗ ΠΥΛΗ

Το Πλέγμα προσφέρει μηχανισμούς για τη δημιουργία και ανακάλυψη υπηρεσιών, μηχανισμούς ασφάλειας κλπ. Ωστόσο πρέπει να υπάρχει ένας τρόπος που να κρύβει την πολυπλοκότητα που εισάγουν όλοι αυτοί οι μηχανισμοί. Η λύση που ακολουθάται πιο συχνά είναι η ανάπτυξη επιστημονικών πυλών (science portals). Οι επιστημονικές πύλες είναι ένα σύνολο από σελίδες στο Web, οι οποίες χρησιμεύουν σαν τη διεπαφή ανάμεσα στο χρήστη και στην εφαρμογή του πλέγματος. Μέσω σελίδων HTML, συλλέγονται οι απαραίτητες πληροφορίες από το χρήστη και προωθούνται στην εφαρμογή. Η εφαρμογή καλείται είτε απευθείας (το πρόγραμμα-πελάτης είναι υλοποιημένο μέσα στις σελίδες χρησιμοποιώντας κάποια τεχνολογία δυναμικών σελίδων) είτε καλεί κάποιο εξωτερικό πρόγραμμα το οποίο προωθεί στη συνέχεια τις πληροφορίες αυτές. Ο χρήστης μπορεί μέσω της πύλης να ελέγχει την πρόοδο της εφαρμογής και τα αποτελέσματα που υπάρχουν. Μια επιστημονική πύλη μπορεί να λειτουργεί ως διεπαφή για παραπάνω από μία εφαρμογές. Το κυριότερο πλεονέκτημα αυτής της λύσης είναι ότι χρησιμοποιείται σαν διεπαφή με το χρήστη το Web, μια τεχνολογία με μεγάλο βαθμό διείσδυσης και γνωστή σε μεγάλο αριθμό χρηστών.

4.1.1 ΑΠΑΙΤΗΣΕΙΣ ΔΙΚΤΥΑΚΗΣ ΠΛΕΓΜΑΤΙΚΗΣ ΠΥΛΗΣ

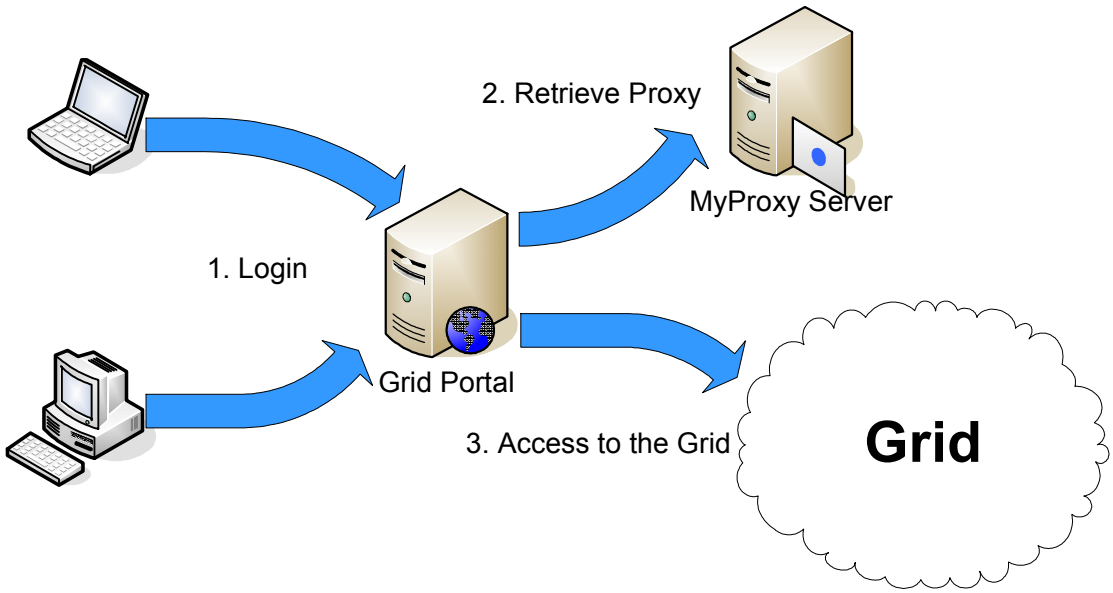
Προκειμένου να διασφαλιστεί η ικανοποίηση των βασικών απαιτήσεων για τη διαδικτυακή πύλη, κρίνεται απαραίτητο να περιγράψουμε τις βασικές αρχές που πρέπει να ακολουθηθούν κατά το σχεδιασμό της.

- Διαλειτουργικότητα. Η διαδικτυακή πύλη πρέπει να σχεδιασθεί με τρόπο τέτοιο ώστε να είναι δυνατή η διασύνδεσή της με απομακρυσμένα συστήματα χωρίς να απαιτούνται επιπλέον ενέργειες από τον τελικό χρήστη. Η χρήση διεπαφών που βασίζονται σε πρότυπα και εργαλεία θα επιτρέψει την ευκολότερη και ταχύτερη ενσωμάτωση υπηρεσιών, χωρίς να επηρεάζει το επίπεδο της ποιότητας και το βαθμό στον οποίο μπορεί να επιτευχθεί παραμετροποίηση του συστήματος
- Ευκολία χρήσης. Η πύλη θα αποτελέσει τη διεπαφή του χρήστη τόσο με την εφαρμογή όσο και με τους υπόλοιπους χρήστες που θα χρησιμοποιήσουν την εφαρμογή. Η ευκολία χρήσης είναι το χαρακτηριστικό που ορίζει πως οι διεπαφές προς τον χρήστη μπορούν να χρησιμοποιηθούν εύκολα. Οι κύριοι παράγοντες που επηρεάζουν την ευκολία χρήσης είναι:

- Πόσο εύκολα μπορεί ένας χρήστης να επιτύχει κάποιες βασικές λειτουργίες κατά την πρώτη φορά που θα χρησιμοποιήσουν τη δικτυακή πύλη.
- Όταν οι χρήστες εξοικειωθούν με τη βασική υποδομή που τους προσφέρεται, πόσο χρόνο μπορούν να εξοικονομήσουν χρησιμοποιώντας την.
- Το χρονικό διάστημα που απαιτείται για αναπροσαρμογή ύστερα από περίοδο μη χρήσης της εφαρμογής.
- Πόσα λάθη κάνουν οι χρήστες, πόσο σημαντικά είναι αυτά τα λάθη και πόσο εύκολη είναι η αποκατάστασή τους.
- Πόσο ευχάριστο είναι το περιβάλλον του χρήστη.
- Ασφάλεια. Η διασφάλιση τόσο των προσωπικών στοιχείων του χρήστη, όσο και των εργασιών που εκτελούνται εκ μέρους του στο πλέγμα πρέπει να είναι βασικό δομικό στοιχείο κατά τη διαδικασία σχεδιασμού της πύλης. Επιπλέον είναι ιδιαίτερα σημαντικό να διαφυλάσσονται οι μηχανισμοί του Πλέγματος και να χρησιμοποιηθεί η ήδη υπάρχουσα υποδομή.
- Δυνατότητες παραμετροποίησης και δόμησης της εφαρμογής σε συστατικά. Συστατικά τα οποία προέρχονται από βιβλιοθήκες ανοιχτού κώδικα έχουν γίνει ιδιαίτερα δημοφιλή. Πολλά προϊόντα βασίζονται εξ'ολοκλήρου σε τεχνολογίες ανοιχτού κώδικα, εκμεταλλευόμενα τη δυνατότητα που προσφέρουν για δόμηση της αρχιτεκτονικής τους από επιμέρους συστατικά τα οποία εκτελούν μια συγκεκριμένη λειτουργία. Το σημαντικότερο πλεονέκτημα αυτών των συστατικών είναι ότι υποστηρίζονται από μεγάλες κοινότητες προγραμματιστών και σχεδιαστών συστημάτων. Η επιτυχής ενσωμάτωση αυτών των συστατικών στην εφαρμογή προϋποθέτει ότι τα επιμέρους συστατικά πρέπει να ικανοποιούν τις λειτουργικές και ποιοτικές απαιτήσεις του συστήματος, ενώ πρέπει να παρέχεται η δυνατότητα διαλειτουργικότητας με τα υπόλοιπα συστατικά του συστήματος.

4.1.2 MyProxy

Ο εξυπηρετητής MyProxy [73] είναι ένα σύστημα διαχείρισης των διαπιστευτηρίων των χρηστών του Πλέγματος. Ο κύριος στόχος του είναι να παρέχει έναν online χώρο διαχείρισης των διαπιστευτηρίων των χρηστών προκειμένου να μπορούν να χρησιμοποιηθούν από εφαρμογές τρίτων όπως δικτυακές πλεγματικές πύλες (grid portals) ή το Globus Toolkit.



Εικόνα 4.4: MyProxy και Δικτυακές Πλεγματικές Πύλες

Το MyProxy μπορεί να χρησιμοποιηθεί με διάφορους τρόπους σε μια δικτυακή πύλη. Ωστόσο η διαδικασία που ακολουθείτε είναι σε γενικές γραμμές η ίδια: οι χρήστες συνδέονται στην πύλη, η οποία αναλαμβάνει να επικοινωνήσει με τον εξυπηρετητή του MyProxy και να λάβει τα κατάλληλα πιστοποιητικά ώστε να αποκτήσει πρόσβαση στους πόρους του Grid εκ μέρους του χρήστη. Η πύλη πρέπει να πιστοποιήσει στον εξυπηρετητή MyProxy ότι έχει την εξουσιοδότηση του χρήστη να λάβει τα πιστοποιητικά. Αυτό μπορεί να επιτευχθεί με δύο τρόπους. Ο ένας είναι ο χρήστης να δώσει στην πύλη ένα συνδυασμό username/password προκειμένου να τα χρησιμοποιήσει η πύλη. Ο δεύτερος τρόπος είναι να χρησιμοποιηθεί κάποιο σύστημα single sign-on, το οποίο προμηθεύει στην πύλη ένα cookie το οποίο μπορεί να χρησιμοποιήσει η πύλη για να πιστοποιήσει τα στοιχεία του χρήστη στον εξυπηρετητή του MyProxy. Παράδειγμα τέτοιου συστήματος είναι το Pubcookie.

4.1.3 PORTLETS

Στην παράγραφο 2.6.4 περιγράψαμε τις γενικές αρχές που προσφέρουν τα portlets. Στην περίπτωση της ανάπτυξης μιας Δικτυακής Πλεγματικής Πύλης, οι λειτουργίες που θα πρέπει να προσφέρονται μέσω των αντίστοιχων portlets είναι οι ακόλουθες:

- Διαχείριση πιστοποιητικών (μέσω MyProxy)
- Διαχείριση απομακρυσμένων αρχείων τόσο μέσω του GridFTP όσο και άλλων συστημάτων διαχείρισης αρχείων
- Portlets για την διαχείριση εργασιών

Επιπλέον, προκειμένου να εξασφαλιστεί ότι ταυτόχρονα με την αξιοποίηση των υπολογιστικών πόρων του Πλέγματος γίνεται και αξιοποίηση του ανθρώπινου δυναμικού που χρησιμοποιεί την Πύλη, κρίνεται αναγκαία και η ύπαρξη των κατάλληλων εργαλείων τα οποία θα επιτρέψουν την καλύτερη επικοινωνία και συνεργασία. Τα portlets αυτά μπορούν να προσφέρουν λειτουργίες όπως forums συζητήσεων, WiKis, ειδήσεις κλπ.

4.2 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ GRID PORTAL

Η δικτυακή πύλη που αναπτύχθηκε βασίστηκε στην πλατφόρμα JBoss Portal. Η πλατφόρμα αυτή προσφέρει την κατάλληλη υποδομή και βιβλιοθήκες λογισμικού προκειμένου να αναπτυχθούν δικτυακές πύλες χρησιμοποιώντας την προδιαγραφή JSR-168 για τα portlets. Τα επιμέρους συστατικά της πλατφόρμας αναπτύσσονται σαν αυτόνομα portlets, προσφέροντας έτσι περισσότερες δυνατότητες παραμετροποίησης τόσο στο επίπεδο παρουσίασης της πύλης όσο και στα συστατικά που αναλαμβάνουν τον έλεγχό της.

Το JBoss Portal προσφέρει μηχανισμούς ασφάλειας οι οποίοι βασίζονται στους ρόλους. Ο κάθε χρήστης έχει έναν ή παραπάνω ρόλους. Ο κάθε ρόλος επιτρέπει ή απαγορεύει στον χρήστη την πρόσβαση σε ένα ή περισσότερα portlets και συστατικά της πύλης.

Όσο αφορά τις λειτουργίες που σχετίζονται με το πλέγμα, χρησιμοποιήθηκαν οι κατάλληλες βιβλιοθήκες που προσφέρονται για την πρόσβαση στα επιμέρους συστατικά του πλέγματος. Τα πιστοποιητικά που χρησιμοποιούνται για τις διάφορες εργασίες στο Πλέγμα προέρχονται από έναν εξυπηρετητή MyProxy.

Για την υποβολή των εργασιών και την επίβλεψη τους χρησιμοποιήθηκε η βιβλιοθήκη Java του EDG [74]. Η αρχική έκδοση είχε βασιστεί στην βιβλιοθήκη gLite [75], ωστόσο προβλήματα ασυμβατότητας με άλλους κόμβους οδήγησαν στην χρησιμοποίηση της βιβλιοθήκης EDG.

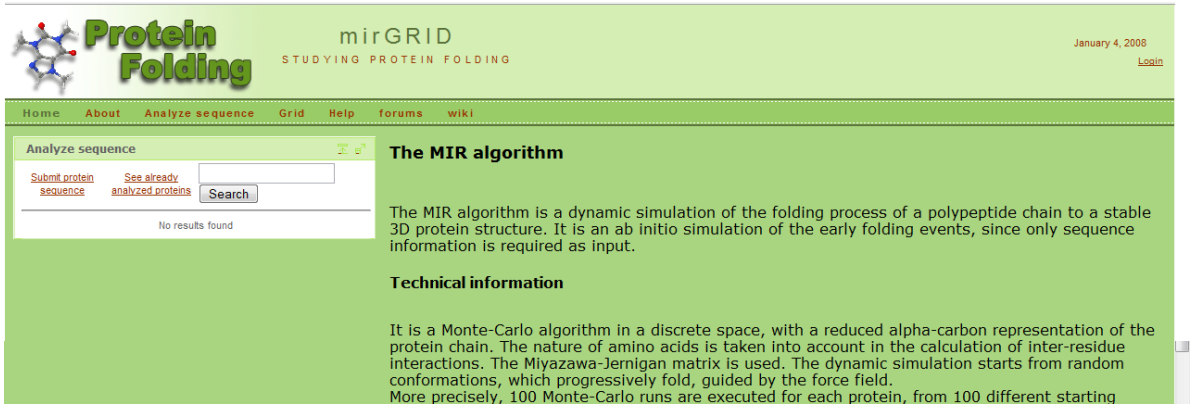
Ένα από τα σημαντικότερα προβλήματα που ανέκυψαν κατά τη διάρκεια της ανάπτυξης της πύλης ήταν ότι τα δεδομένα εισόδου για τις εργασίες προέρχονται από πολλές διαφορετικές πηγές και χρήστες. Ωστόσο ελάχιστοι από τους χρήστες έχουν έγκυρα πιστοποιητικά για το Πλέγμα, με αποτέλεσμα να μην μπορούν οι ίδιοι να υποβάλουν εργασίες στο Πλέγμα λόγω της πολιτικής που ακολουθείται από τους αρμόδιους οργανισμούς. Προκειμένου να ξεπεραστεί αυτό το πρόβλημα αναπτύχθηκε ένας μηχανισμός ο οποίος επιτρέπει στους απλούς χρήστες που δεν έχουν κάποιο πιστοποιητικό να δηλώσουν κάποια δεδομένα τα οποία έχουν κάποιο ενδιαφέρον για αυτούς. Τα δεδομένα αυτά μπορούν να προέρχονται είτε από ευρέως γνωστές βάσεις δεδομένων πρωτεϊνών (τις PDB [77] και SwissProt [78]) είτε σε μορφή κειμένου από τους χρήστες. Το portlet που αναλαμβάνει αυτή την εργασία

αποθηκεύει στατιστικά για τις αιτήσεις των επιμέρους πρωτεϊνών, τα οποία μπορούν να δουν οι χρήστες που διαθέτουν κάποια πιστοποιητικά και να επιλέξουν κάποιες πρωτεΐνες ώστε να τις υποβάλουν στο Πλέγμα.

Τέλος, επιπλέον μέριμνα υπάρχει σχετικά με την ύπαρξη εργαλείων τα οποία θα επιτρέψουν την ανάπτυξη μιας κοινότητας γύρω από την εφαρμογή. Τα εργαλεία αυτά περιλαμβάνουν:

- Wiki, το οποίο επιτρέπει την τεκμηρίωση των λειτουργιών της πύλης καθώς και ένα μέρος όπου μπορούν να αποθηκευτούν τα συμπεράσματα από τις εργασίες που τρέχουν στο Πλέγμα.
- Forums συζητήσεων προκειμένου να είναι πιο εύκολη η συζήτηση πάνω σε θέματα σχετικά τόσο με την ανάπτυξη της εφαρμογής όσο και τα επιμέρους αποτελέσματα.

Στην Εικόνα 4.5 φαίνεται η κεντρική σελίδα της δικτυακής πύλης.



Εικόνα 4.5: Κεντρική σελίδα δικτυακής πύλης

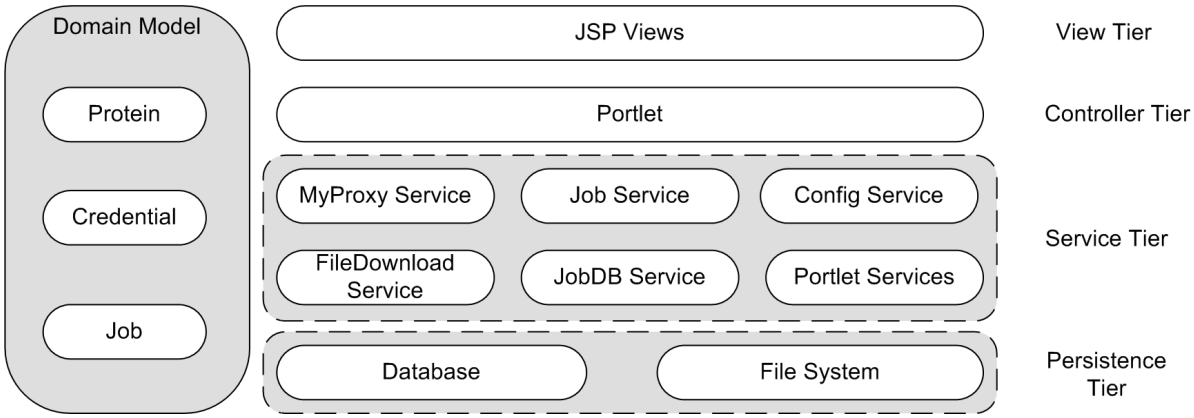
4.3 Αρχιτεκτονικη του Grid Portlet

Η αρχιτεκτονική που χρησιμοποιήθηκε εσωτερικά για το portlet είναι μια αρχιτεκτονική πολλών επιπέδων (Εικόνα 4.6). Ο διαχωρισμός σε επίπεδα βοηθά στην ευκολότερη ανάπτυξη του portlet αλλά και της δοκιμής των επιμέρους συστατικών που το αποτελούν. Τα επίπεδα αυτά είναι ονομαστικά:

- Επίπεδο οπτικοποίησης: λειτουργεί ως η διεπαφή προς το χρήστη. Αναλαμβάνει την εμφάνιση των δεδομένων και προωθεί τις εντολές του χρήστη στο portlet.
- Επίπεδο ελεγκτή (controller): Αναλαμβάνει το συντονισμό των συστατικών που αποτελούν την εφαρμογή. Ελέγχει τα δεδομένα που εισάγει ο χρήστης και ανάλογα με την τρέχουσα κατάσταση χρησιμοποιεί τις κατάλληλες υπηρεσίες που προσφέρονται από το επίπεδο υπηρεσιών.
- Επίπεδο υπηρεσιών: Περιέχει τις λειτουργίες που απαιτούνται για την υλοποίηση της εφαρμογής. Στην περίπτωσή του portlet οι υπηρεσίες αυτές συμπεριλαμβάνουν

υπηρεσίες διαχείρισης πιστοποιητικών μέσω του εξυπηρετητή MyProxy, υπηρεσίες διαχείρισης εργασιών (π.χ. αποστολή, έλεγχος κατάστασης, ακύρωση και λήψη αποτελεσμάτων), υπηρεσίες αποθήκευσης δεδομένων στη βάση δεδομένων, υπηρεσίες διαμόρφωσης ρυθμίσεων και υπηρεσίες σχετικά με το Portlet.

- Επίπεδο αποθήκευσης: Προσφέρει τις απαραίτητες λειτουργίες για την αποθήκευση των δεδομένων και των αποτελεσμάτων. Τα δεδομένα που αποτελούν τα αντικείμενα που περιγράφουν το μοντέλο της εφαρμογής αποθηκεύονται σε βάση δεδομένων, ενώ τα αρχεία που περιέχουν τα αποτελέσματα αποθηκεύονται σε ένα σύστημα αρχείων.
- Το μοντέλο της εφαρμογής: Αποτελείται από αντικείμενα τα οποία αναπαριστούν διακριτές οντότητες, όπως για παράδειγμα η πρωτεΐνη που αντιστοιχεί σε κάθε εργασία, το όνομα του πιστοποιητικού που χρησιμοποιήθηκε, και χρήσιμες πληροφορίες σχετικά με τις εργασίες που στάλθηκαν στο Πλέγμα προς εκτέλεση, όπως χρόνος μετάβασης σε κάθε κατάσταση, κόμβος στον οποίο εκτελέστηκε η εργασία και η τελική κατάσταση. Μιας και το μοντέλο αυτό είναι απαραίτητο από όλα τα άλλα επίπεδα, το επίπεδο αυτό είναι κατακόρυφο στην εφαρμογή μας.



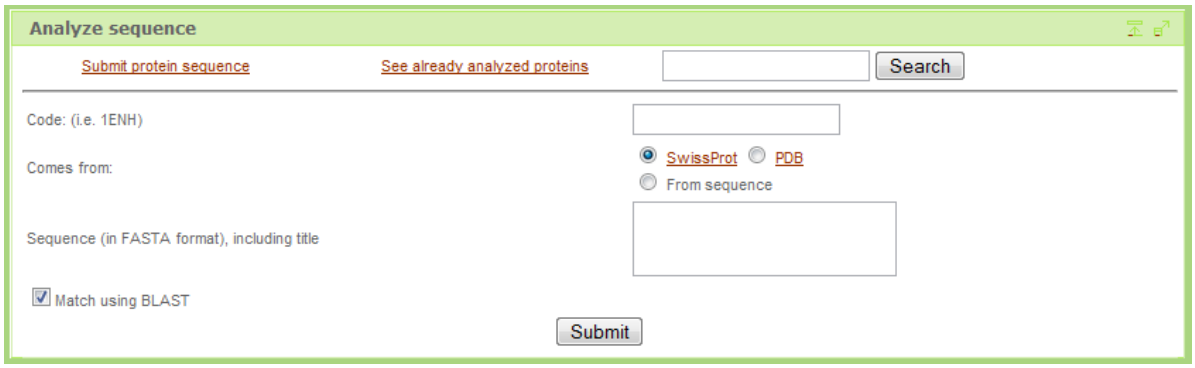
Εικόνα 4.6: Εσωτερική αρχιτεκτονική του Portlet

4.4 ΤΕΛΙΚΗ ΜΟΡΦΗ ΔΙΚΤΥΑΚΗΣ ΠΥΛΗΣ

Σε αυτή την παράγραφό γίνεται παρουσίαση των παραπάνω συστατικών της τελικής μορφής της δικτυακής πύλης όπως αυτά εμφανίζονται στον φυλλομετρητή ιστού του χρήστη, με επεξήγηση των λειτουργιών που μπορεί να πραγματοποιήσει ο χρήστης.

4.4.1 ΥΠΟΒΟΛΗ ΑΙΤΗΣΕΩΝ ΠΡΩΤΕΪΝΩΝ

Στην Εικόνα 4.7 φαίνεται η φόρμα υποβολής αίτησης για εκτέλεση προσομοίωσης στην δικτυακή πύλη για μια συγκεκριμένη αλυσίδα αμινοξέων.



Εικόνα 4.7: Φόρμα υποβολής αίτησης για προσθήκη πρωτεΐνης

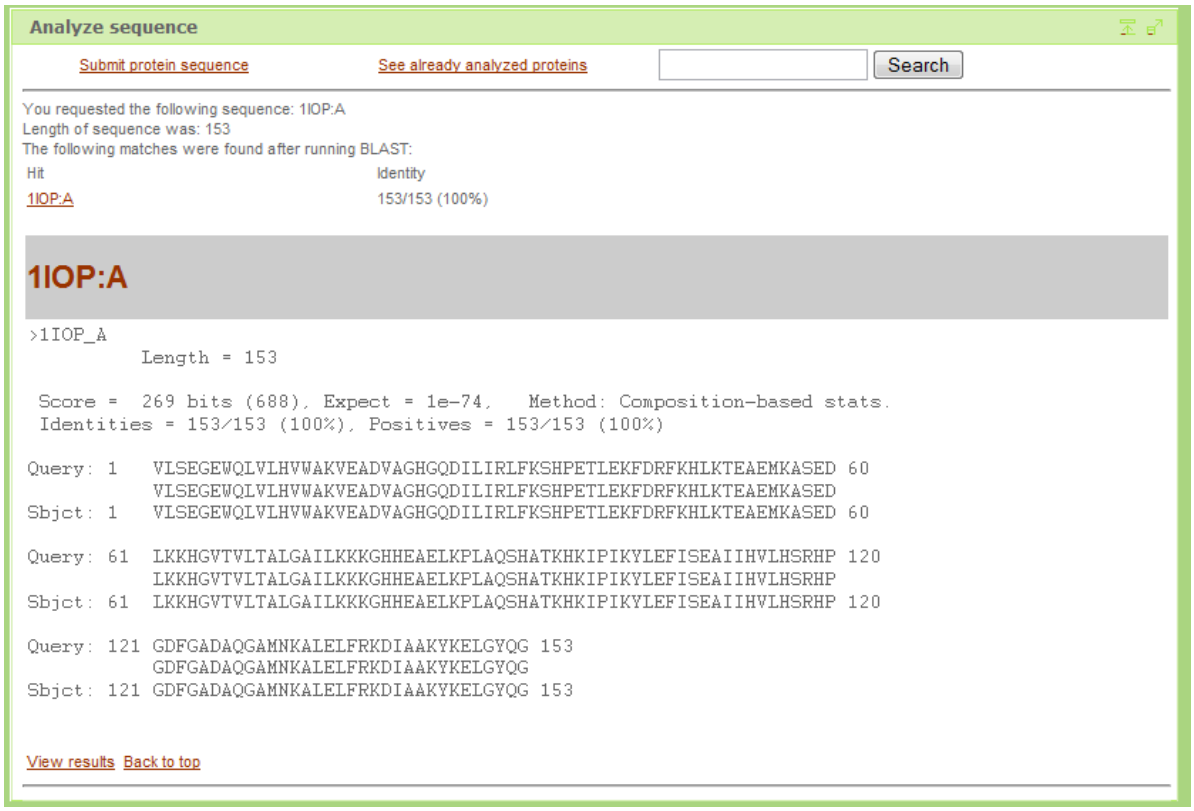
Ο χρήστης μπορεί να εισάγει την πρωτεΐνη της επιλογής του με τρεις διαφορετικούς τρόπους:

- Βάσει του κωδικού SwissProt. Σε αυτή την περίπτωση το portlet αναλαμβάνει να συνδεθεί με τη βάση δεδομένων της SwissProt και να βρει τις κατάλληλες πληροφορίες.
- Βάσει του κωδικού PDB. Αντίστοιχα το portlet συνδέεται με τη βάση δεδομένων της PDB .
- Βάσει του κωδικού και της ακολουθίας σε μορφή FASTA

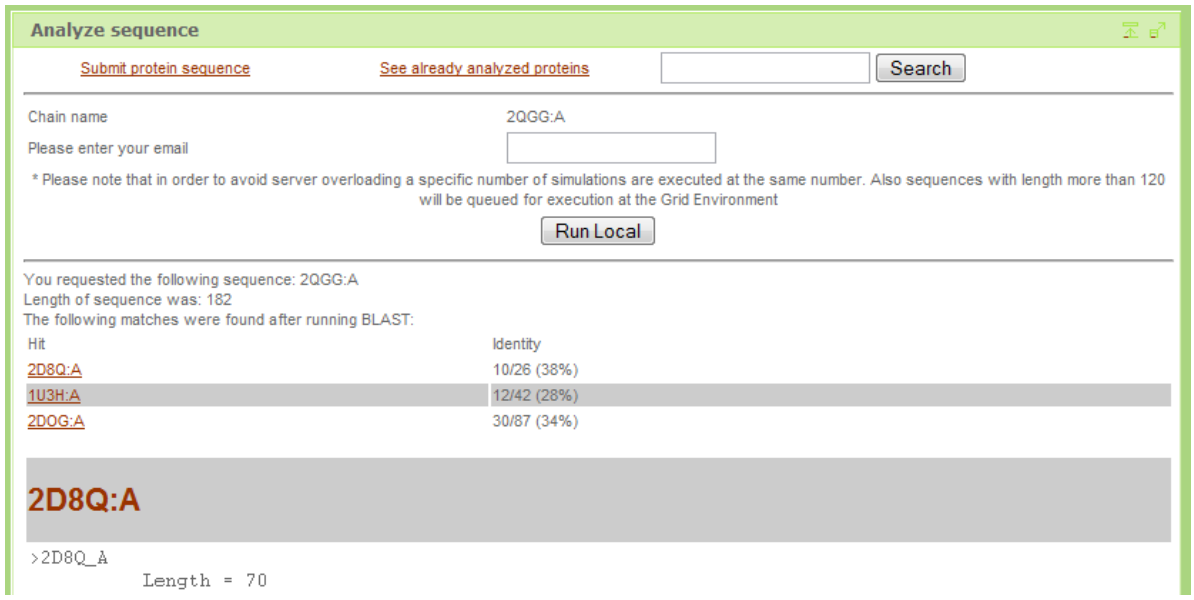
Προτού γίνει η αίτηση για υποβολή της ακολουθίας ο χρήστης μπορεί να επιλέξει αν θα γίνει έλεγχος για την ύπαρξη της ίδιας ή παρόμοιας ακολουθίας χρησιμοποιώντας το εργαλείο BLAST [76]. Σε περίπτωση που υπάρχει ήδη η ακολουθία, ο χρήστης μπορεί να δει τα αποτελέσματα (Εικόνα 4.8). Αντίθετα εμφανίζεται μια λίστα με τις ακολουθίες που υπάρχει μερική ταύτιση, μαζί με τα αποτελέσματά τους. Ο χρήστης έχει τη δυνατότητα να ζητήσει να τρέξει η προσομοίωση τοπικά. Αν το μήκος της ακολουθίας είναι ως ένα συγκεκριμένο κατώφλι, τότε η ακολουθία μπαίνει σε μια ουρά για να τρέξει τοπικά, και ο χρήστης ειδοποιείται μέσω e-mail ότι η προσομοίωση ολοκληρώθηκε (Εικόνα 4.9). Σε αντίθετη περίπτωση, όπου το μήκος της ακολουθίας είναι αρκετά μεγάλο, η ακολουθία μπαίνει σε μια λίστα για να υποβληθεί στο Πλέγμα. Το όφελος από αυτόν το μηχανισμό είναι ότι για μικρές ακολουθίες – που αντιστοιχούν και σε προσομοιώσεις με μικρό χρόνο εκτέλεσης – τα αποτελέσματα είναι πιο άμεσα προσβάσιμα, καθώς ο χρόνος αναμονής πριν

το τρέξιμο της εργασίας στο Πλέγμα είναι αρκετά μεγαλύτερος από το χρόνο εκτέλεσης της προσομοίωσης.

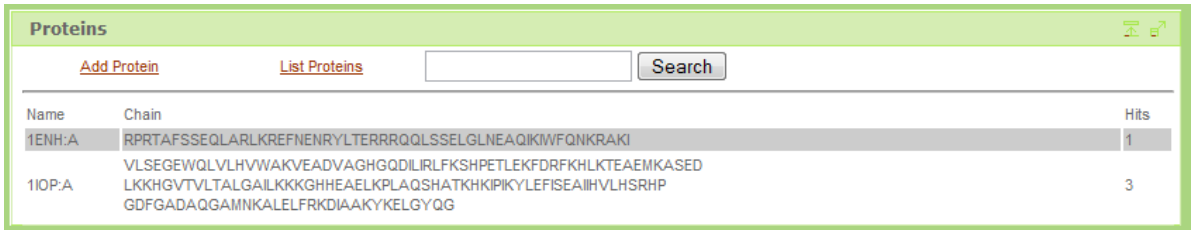
Επιπλέον, ο χρήστης μπορεί είτε να αναζητήσει μια πρωτεΐνη στις ήδη υπάρχουσες βάσει του κωδικού της χρησιμοποιώντας το πεδίο αναζήτησης είτε να δει μια λίστα με τις πρωτεΐνες οι οποίες έχουν ζητηθεί (Εικόνα 4.10).



Εικόνα 4.8: Έλεγχος με BLAST με απόλυτη ταύτιση και αποτελέσματα



Εικόνα 4.9: Έλεγχος με BLAST με μερική ταύτιση και αποτελέσματα



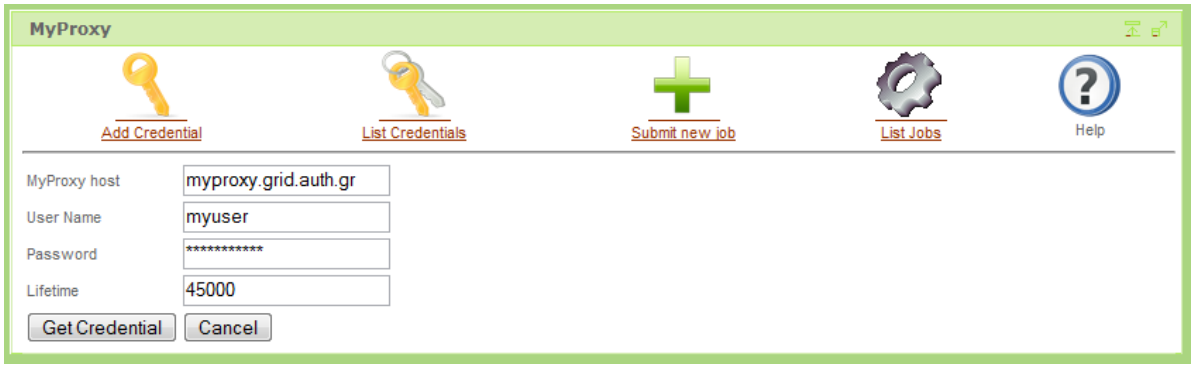
Εικόνα 4.10: Λίστα πρωτεϊνών

4.4.2 ΔΙΑΧΕΙΡΙΣΗ ΠΙΣΤΟΠΟΙΗΤΙΚΩΝ ΚΑΙ ΕΡΓΑΣΙΩΝ ΧΡΗΣΤΗ

Η διαχείριση των πιστοποιητικών του χρήστη καθώς και των εργασιών που αυτός έχει υποβάλλει στο Πλέγμα γίνεται μέσα από το Grid Portlet.

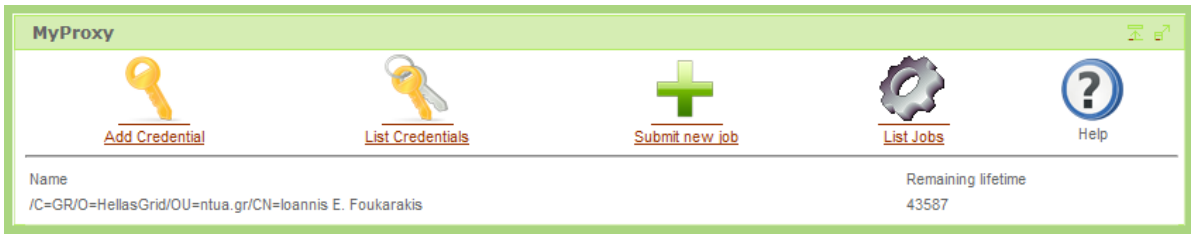
Η πρώτη λειτουργία που μπορεί να εκτελέσει ο χρήστης είναι να λάβει ένα πιστοποιητικό από κάποιο εξυπηρετητή MyProxy. Τα στοιχεία που πρέπει να εισάγει ο χρήστης είναι τα ακόλουθα:

- Η διεύθυνση του εξυπηρετητή MyProxy
- Το όνομα του χρήστη (username)
- Ο κωδικός του πιστοποιητικού
- Η διάρκεια (σε δευτερόλεπτα) για την οποία το πιστοποιητικό θα είναι διαθέσιμο στην πύλη.



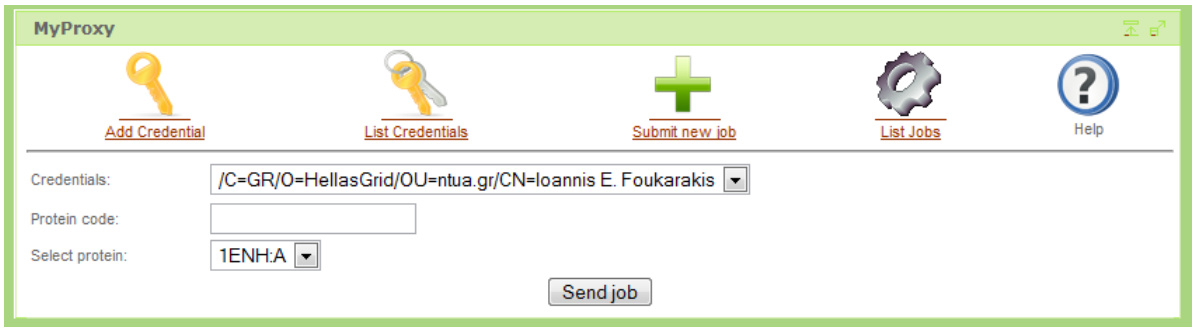
Εικόνα 4.11: Εισαγωγή πιστοποιητικών από εξυπηρετητή MyProxy

Ο χρήστης μπορεί να δει όποτε επιθυμεί μια λίστα με τα πιστοποιητικά που είναι ενεργά (Εικόνα 4.12).



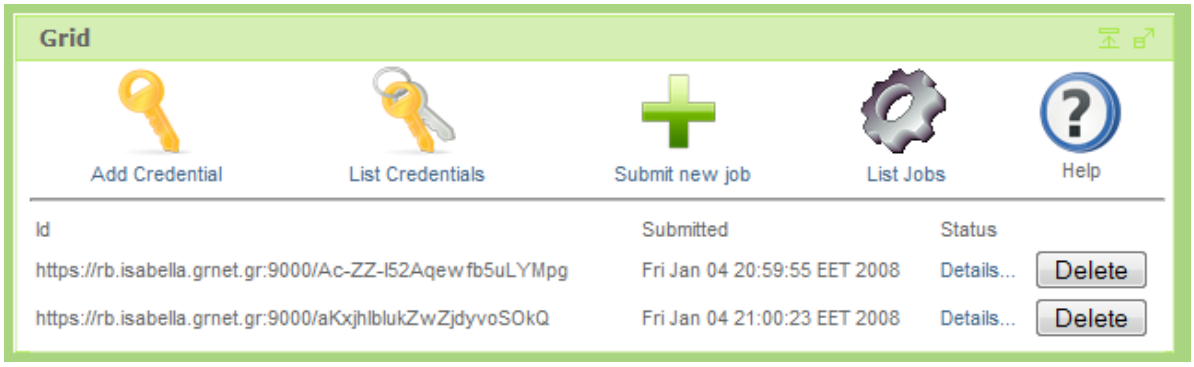
Εικόνα 4.12: Λίστα ενεργών πιστοποιητικών

Μέσω της επιλογής "Submit new job" είναι δυνατή η υποβολή μιας νέας εργασίας στο πλέγμα. Ο χρήστης επιλέγει πιο πιστοποιητικό θα χρησιμοποιήσει καθώς και τον κωδικό κάποιας πρωτεΐνης είτε πληκτρολογώντας τον είτε επιλέγοντάς τον από μια λίστα (Εικόνα 4.13).



Εικόνα 4.13: Φόρμα υποβολής εργασιών στο Πλέγμα.

Τέλος είναι δυνατή η επίβλεψη της κατάστασης των εργασιών που έχουν υποβληθεί στο Πλέγμα. Κάνοντας κλικ στο «List Jobs» εμφανίζεται μια λίστα με τις εργασίες που έχουν εκτελεστεί στο Πλέγμα (Εικόνα 4.14), ενώ με κλικ στον κωδικό της εργασίας εμφανίζονται αναλυτικές πληροφορίες σχετικά με την εργασία (Εικόνα 4.15).



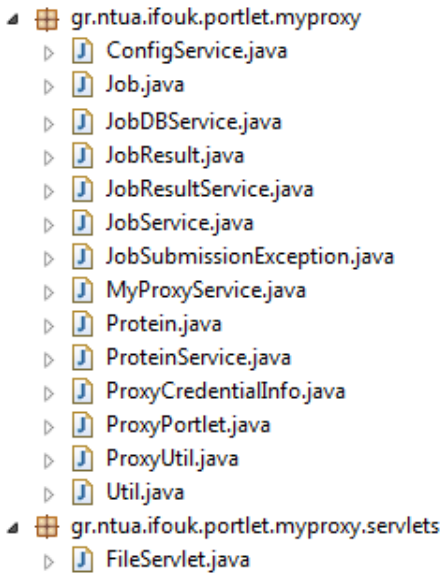
Εικόνα 4.14: Λίστα εργασιών που έχει υποβάλει ο χρήστης στο Πλέγμα.

Εικόνα 4.15: Στοιχεία εργασίας που έχει υποβληθεί στο Πλέγμα

4.5 ΠΕΡΙΓΡΑΦΗ ΚΩΔΙΚΑ ΣΥΣΤΑΤΙΚΩΝ ΔΙΚΤΥΑΚΗΣ ΠΥΛΗΣ

Προκειμένου να ικανοποιηθούν οι απαιτήσεις που αναφέρθηκαν παραπάνω ήταν αναγκαία η ανάπτυξη δύο portlets. Το πρώτο είναι το MyProxy Portlet το οποίο είναι η διεπαφή των χρηστών με το Πλέγμα. Αναλαμβάνει τη διαχείριση των πιστοποιητικών και επιτρέπει στους χρήστες που είναι εξοικειωμένοι με τις βασικές έννοιες του Πλέγματος να υποβάλλουν εργασίες και να τις διαχειριστούν. Το δεύτερο είναι το Protein Portlet και στόχος του είναι να επιτρέψει σε χρήστες που δεν έχουν κάποιο πιστοποιητικό διαθέσιμο να υποβάλλουν αιτήσεις για ακολουθίες οι οποίες τους ενδιαφέρουν. Στην παράγραφο αυτό θα περιγράψουμε τις σημαντικότερες κλάσεις Java που χρησιμοποιήθηκαν για την ανάπτυξη αυτών των portlets.

4.5.1 MyProxy Portlet



Το MyProxy Portlet αποτελείτε από ένα σύνολο κλάσεων που επιτρέπουν τη διαχείριση προσωρινών πιστοποιητικών από το πλέγμα, την επίβλεψη των εργασιών που έχουν υποβληθεί, καθώς και την διαχείριση των αποτελεσμάτων που έχουν αποθηκευτεί στη βάση. Οι κλάσεις που χρησιμοποιήθηκαν είναι οι ακόλουθες:

ConfigService

Η κλάση αυτή διαβάζει κατά την εκκίνηση του εξυπηρετητή πληροφορίες από ένα αρχείο ρυθμίσεων σε μορφή .properties. Οι πληροφορίες αυτές είναι κυρίως διαδρομές στο δίσκο και δείχνουν τον κατάλογο προσωρινής αποθήκευσης αρχείων, τον κατάλογο αποθήκευσης των αποτελεσμάτων κλπ.

Job

Πρόκειται για ένα Java Bean κάθε στιγμιότυπο του οποίου αποθηκεύει πληροφορίες για μια εργασία που έχει υποβληθεί στο Πλέγμα. Είναι ιδιαίτερα σημαντική κλάση μιας και τα Java Beans αυτά επιτρέπουν στον χρήστη να επιβλέπουν της εργασίες τους σε διαφορετικές συνδέσεις τους στην πύλη. Οι πληροφορίες που αποθηκεύονται σε αυτή την κλάση είναι το id της εργασίας, ο χρήστης ο οποίος υπέβαλλε την εργασία και στοιχεία για την πορεία της εργασίας (ημερομηνία υποβολής, αν έχει ολοκληρώσει την εκτέλεσή της και αν έχουν ληφθεί τα αποτελέσματα από το Πλέγμα.

JobDBService

Η κλάση αυτή προσφέρει υπηρεσίες σχετικές με την αποθήκευση των δεδομένων της κλάσης Job. Χρησιμοποιεί τη βιβλιοθήκη Hibernate 3 [72] προκειμένου να επιτρέπει αποθήκευση, ανάκτηση, ενημέρωση και διαγραφή των αντικειμένων αυτών χωρίς να είναι απαραίτητο να είναι εκ των προτέρων γνωστή η βάση δεδομένων. Την βιβλιοθήκη αυτή χρησιμοποιούν και οι υπόλοιπες κλάσεις που προσφέρουν υπηρεσίες αποθήκευσης δεδομένων.

JobService

Η κλάση αυτή αναλαμβάνει να προσφέρει τις απαραίτητες υπηρεσίες για την υποβολή των εργασιών στο Πλέγμα. Οι συνήθεις λειτουργίες που γίνονται στο Πλέγμα μέσω της

γραμμής εντολής του UI node μπορούν να πραγματοποιηθούν προγραμματιστικά χρησιμοποιώντας τις μεθόδους αυτής της τάξης. Έτσι επιτρέπεται η υποβολή μιας εργασίας στο Πλέγμα, ο έλεγχος της κατάστασής της, η ακύρωσή της, η εύρεση των κόμβων στους οποίους μπορεί αυτή να εκτελεστεί και η λήψη των αποτελεσμάτων της εκτέλεσης. Οι λειτουργίες αυτές γίνονται χρησιμοποιώντας το EDG API.

JobResult

Ένα Java Bean το οποίο κρατάει πληροφορίες για τα αποτελέσματα που υπάρχουν αποθηκευμένα στην πύλη. Οι πληροφορίες αυτές είναι μια αναφορά στα στοιχεία της εργασίας που έτρεξε στο Πλέγμα (αν αυτή προήλθε από αυτό), την ακολουθία η οποία αποτέλεσε την είσοδο για την προσομοίωση καθώς και τη διαδρομή στο δίσκο που είναι αποθηκευμένο.

JobResultService

Αναλαμβάνει τις εργασίες της αποθήκευσης, ανάκτησης ενημέρωσης και διαγραφής των αντικειμένων της κλάσης JobResult από τη βάση δεδομένων.

JobSubmissionException

Γενική κλάση exception για περιπτώσεις που δημιουργείται κάποιο πρόβλημα στη διαδικασία υποβολής εργασίας στο Πλέγμα.

MyProxyService

Αναλαμβάνει τη διαχείριση των προσωρινών πιστοποιητικών του χρήστη. Πιο αναλυτικά, αναλαμβάνει να συνδεθεί στον εξυπηρετητή MyProxy και να λάβει το προσωρινό πιστοποιητικό, το οποίο κρατάει στη μνήμη μέχρι αυτό να λήξει. Στη συνέχεια, όταν είναι απαραίτητο για κάποια εργασία, ο χρήστης μπορεί να χρησιμοποιήσει κάποιο από αυτά τα πιστοποιητικά.

Protein

Java Bean για την αποθήκευση των δεδομένων κάθε ακολουθίας που υπάρχει στην Πύλη, είτε αυτή έχει εκτελεσθεί είτε όχι. Συγκρατεί πληροφορίες όπως ο τίτλος και ο κωδικός της ακολουθίας, την ίδια την ακολουθία και το πόσοι χρήστες έχουν ζητήσει την εκτέλεσή της.

ProteinService

Αναλαμβάνει τις εργασίες της αποθήκευσης, ανάκτησης ενημέρωσης και διαγραφής των αντικειμένων της κλάσης Protein από τη βάση δεδομένων.

ProxyCredentialInfo

Java Bean το οποίο κρατάει τμήμα των πληροφοριών που περιγράφονται στο προσωρινό πιστοποιητικό του χρήστη.

ProxyPortlet

Ο πυρήνας του portlet. Η κλάση αυτή αναλαμβάνει να δέχεται τις αιτήσεις από τον χρήστη, να πραγματοποιεί τις κατάλληλες ενέργειες και να εμφανίζει το αποτέλεσμα. Υλοποιεί τον ελεγκτή (controller) του portlet. Ανάλογα με την είσοδο του χρήστη (αιτήσεις λήψεις προσωρινού πιστοποιητικού, εντολές διαχείρισης εργασιών) και την τρέχουσα κατάσταση (αν υπάρχει κάποιο προσωρινό πιστοποιητικό διαθέσιμο, εργασίες οι οποίες έχουν υποβληθεί στο Πλέγμα κλπ) , καλεί τις κατάλληλες μεθόδους των κλάσεων – υπηρεσιών (ConfigService, JobService, JobDBService, ProteinService και JobResultService) προκειμένου να εκτελέσει τις λειτουργίες που επιθυμεί ο χρήστης. Τα αποτελέσματα προωθούνται σε μία σελίδα JSP όπου και γίνεται η απεικόνισή τους.

ProxyUtil

Προσφέρει βοηθητικές λειτουργίες για την επεξεργασία των προσωρινών πιστοποιητικών.

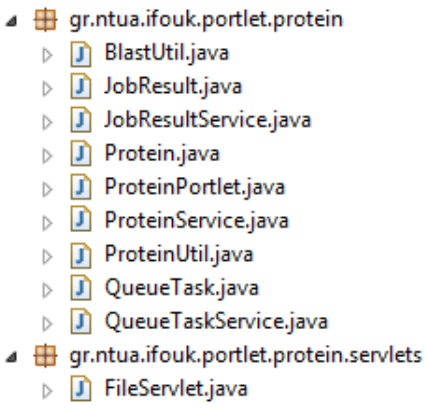
Util

Περιέχει μεθόδους με βοηθητικές λειτουργίες (κυρίως για διαχείριση αρχείων) που δεν ανήκουν σε κάποια άλλη οντότητα του portlet.

FileServlet

Java Servlet που επιτρέπει στον χρήστη την λήψη αρχείων τα οποία βρίσκονται σε συγκεκριμένο φάκελο εκτός του εξυπηρετητή ιστού. Χρησιμοποιείται για να μπορούν οι χρήστες να λάβουν τα αποτελέσματα των προσομοιώσεων.

4.5.2 PROTEIN PORTLET



Το portlet αυτό περιέχει τον κώδικα που επιτρέπει σε απλούς χρήστες που δεν έχουν τη βασική εξοικίωση με τις έννοιες του Πλέγματος ή δεν διαθέτουν κάποιο πιστοποιητικό να δουν τα εως τώρα αποτελέσματα και να ζητήσουν την εκτέλεση κάποιας συγκεκριμένης ακολουθίας. Πολλές από τις τάξεις που έχουν χρησιμοποιηθεί είναι παρόμοιες με αυτές του MyProxy Portlet, μιας και περιγράφουν τα ίδια δεδομένα.

BlastUtil

Κλάση που επιτρέπει την εκτέλεση του BLAST προκειμένου να πραγματοποιείται έλεγχος για το αν μια ακολουθία ή παρόμοιά της υπάρχει ήδη στη βάση δεδομένων. Αναλαμβάνει

την εκτέλεση του προγράμματος και την ανανέωση της εσωτερικής βάσης δεδομένων που χρησιμοποιεί το BLAST.

JobResult

Όπως και στο MyProxy Portlet.

JobResultService

Όπως και στο MyProxy Portlet.

Protein

Όπως και στο MyProxy Portlet.

ProteinService

Όπως και στο MyProxy Portlet.

ProteinPortlet

Ο ελεγκτής (controller) του portlet. Χρησιμοποιώντας τις υπηρεσίες που προσφέρονται από τις άλλες κλάσεις επιτρέπει στον χρήστη να δει τις ακολουθίες που έχουν υποβληθεί, τα αποτελέσματα τους, αλλά και να υποβάλλει καινούργιες ακολουθίες. Τα αποτελέσματα κάθε εντολής του χρήστη προωθούνται σε σελίδες JSP όπου και εμφανίζονται.

ProteinUtil

Προσφέρει βασικές λειτουργίες για την λήψη ακολουθιών από απομακρυσμένες δικτυακές βάσεις δεδομένων όπως η PDB και η SwissProt, καθώς και για την μετατροπή τους από κείμενο σε μορφή που καταλαβαίνει το Portlet.

QueueTask

Java Bean που αποθηκεύει τις πληροφορίες για μια εργασία που θα εκτελεσθεί τοπικά στον εξυπηρετητή, Οι πληροφορίες αυτές είναι η ακολουθία που θα χρησιμοποιηθεί σαν είσοδος, το email του χρήστη που έκανε την αίτηση προκειμένου να του αποσταλθεί ειδοποίηση για τα αποτελέσματα και η ημερομηνία υποβολής της αίτησης.

QueueTaskService

Αναλαμβάνει τις εργασίες της αποθήκευσης, ανάκτησης ενημέρωσης και διαγραφής των αντικειμένων της κλάσης :QueueTask από τη βάση δεδομένων.

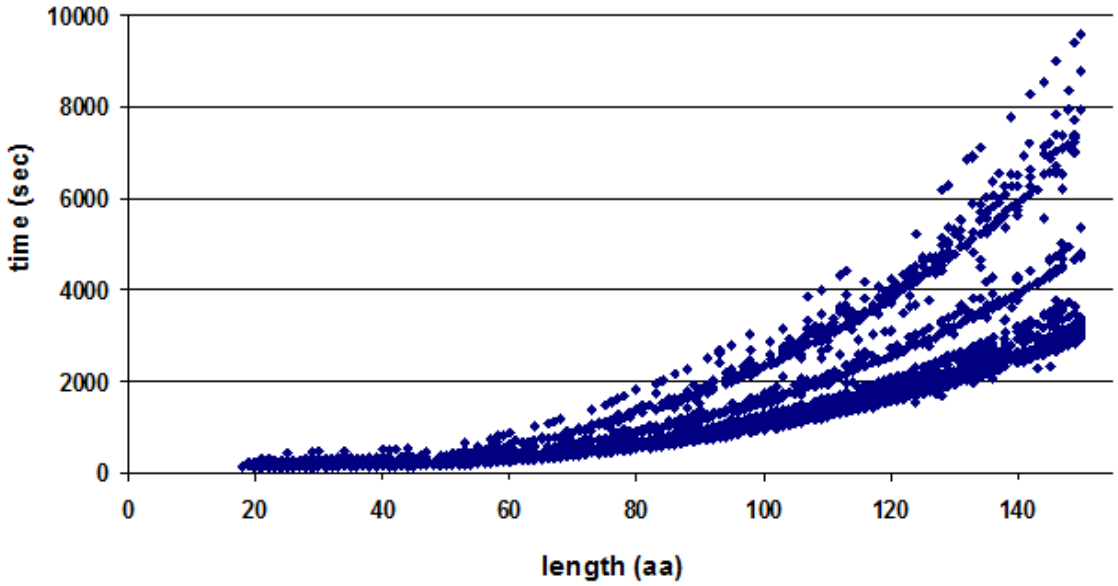
FileServlet

Όπως και στο MyProxy Portlet.

4.6 ΑΠΟΤΕΛΕΣΜΑΤΑ

Χρησιμοποιώντας την εφαρμογή που περιγράψαμε παραπάνω πραγματοποιήθηκαν προσομοιώσεις για περίπου 6000 ακολουθίες αμινοξέων. Οι ακολουθίες αυτές προήλθαν από τη βάση δεδομένων PDB, και αποτελούν αντιπροσωπευτικό δείγμα της. Ο χρόνος

εκτέλεσης για αυτόν τον αριθμό ακολουθιών σε έναν υπολογιστή με ισχύ παρόμοια με αυτούς που είναι διαθέσιμοι στο Πλέγμα είναι περίπου 3,4 μήνες. Με την αποστολή των ακολουθιών στο Πλέγμα επιτεύχθηκε η ολοκλήρωση της εκτέλεσης της προσομοίωσης σε 3,4 εβδομάδες.



Εικόνα 4.16: Χρόνος εκτέλεσης προσομοίωσης συναρτήσει του μήκους της ακολουθίας

Ένα από τα σημαντικότερα δεδομένα είναι ο χρόνος εκτέλεσης του αλγόριθμου για την κάθε ακολουθία. Όπως φαίνεται από τα αποτελέσματα, οι επιμέρους ακολουθίες μπορούν να κατηγοριοποιηθούν σε ομάδες βάσει του χρόνου εκτέλεσής τους. Η κατηγοριοποίηση αυτή έχει ιδιαίτερη σημασία μιας και μπορεί να αναδείξει κάποιες κοινές βιολογικές ιδιότητες.

5 ΠΛΑΤΦΟΡΜΑ ΜΟΝΤΕΛΟΚΕΝΤΡΙΚΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΓΙΑ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

5.1 ΠΡΟΒΛΗΜΑΤΑ ΚΑΤΑ ΤΗΝ ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΩΝ ΛΟΓΙΣΜΙΚΟΥ

Ο κλασικός κύκλος ανάπτυξης μιας εφαρμογής λογισμικού είναι συνήθως ο ακόλουθος:

1. Κατανόηση της έννοιας του προβλήματος και συλλογή απαιτήσεων
2. Ανάλυση του προβλήματος και περιγραφή λειτουργικότητας
3. Σχεδιασμός
4. Ανάπτυξη κώδικα
5. Δοκιμή
6. Εγκατάσταση

Κατά τις τρεις πρώτες φάσεις παράγονται κείμενα και διαγράμματα που περιγράφουν το πρόβλημα. Το κείμενο και οι εικόνες που παράγονται αποτελούν την αποτύπωση στο χαρτί του προβλήματος. Μία από τις πιο δημοφιλείς γλώσσες για την περιγραφή του προβλήματος είναι η Unified Modeling Language (UML), η οποία προσφέρει διαγράμματα για μελέτες περίπτωσης, ιεραρχίας τάξεων κλπ. Παρόλο που αυτά τα κείμενα αποτελούν ένα πρότυπο για την ανάπτυξη του κώδικα, πολλές φορές υπάρχουν προβλήματα στην μετατροπή τους σε κώδικα. Επιπλέον, στην περίπτωση που πραγματοποιηθεί κάποια αλλαγή στη περιγραφή του προβλήματος, η απόσταση από τα διαγράμματα είναι συχνά μεγάλη, με αποτέλεσμα οι αλλαγές να γίνονται απευθείας στον κώδικα για να μειωθεί ο χρόνος ανάπτυξης.

Το δεύτερο πρόβλημα που προκύπτει από την παραπάνω διαδικασία ανάπτυξης εφαρμογών λογισμικού είναι ότι συνήθως η τελική εφαρμογή στηρίζεται σε μια συγκεκριμένη τεχνολογία ή γλώσσα προγραμματισμού. Συχνά είναι επιθυμητή η μετάβαση σε μια νέα τεχνολογία ή γλώσσα (π.χ. μετάβαση από RMI σε Web Services). Ωστόσο αυτό απαιτεί τις περισσότερες φορές να αναπτυχθεί από την αρχή η εφαρμογή.

Μία λύση για τα παραπάνω προβλήματα ήταν ο σχεδιασμός των εφαρμογών λογισμικού να σταματήσει να είναι μονολιθικός, και να ακολουθηθεί μια αρχιτεκτονική όπου η κάθε εφαρμογή αποτελείται από ανεξάρτητα συστατικά. Τα συστατικά αυτά είναι ευκολότερο να αντικατασταθούν, ενώ είναι δυνατό να επαναχρησιμοποιηθούν. Ωστόσο αυτή η προσέγγιση δημιούργησε την ανάγκη να οριστεί ο τρόπος με τον οποίο επικοινωνούν τα συστατικά. Αποτέλεσμα είναι η ύπαρξη πολλών διαφορετικών προδιαγραφών όπως CORBA, SOAP, REST κλπ.

Στα προηγούμενα κεφάλαια εξετάσαμε διάφορες τεχνολογίες λογισμικού για κατανεμημένα συστήματα, οι οποίες μπορούν να χρησιμοποιηθούν για την επίλυση πολλών διαφορετικών προβλημάτων. Ωστόσο όλες έχουν ένα κοινό σημείο. Για να μπορέσει να αναπτυχθεί η κάθε εφαρμογή χρειάζεται ο σχεδιασμός του μοντέλου βάσει του οποίου θα λειτουργεί. Για παράδειγμα, οι κινητοί πράκτορες χρησιμοποίησαν το μοντέλο αφέντη-εργάτη, ενώ για κάθε εφαρμογή ιστού απαιτείται η ύπαρξη ενός μοντέλου που αναπαριστά τα δεδομένα, προκειμένου να μπορούν να αναπτυχθούν τα επιμέρους συστατικά, όπως π.χ. το σχήμα της βάσης δεδομένων και οι φόρμες εισαγωγής των δεδομένων.

Το μεγαλύτερο πρόβλημα σε όλες τις παραπάνω περιπτώσεις είναι ότι παρόλο που ο σχεδιασμός των συστημάτων γίνεται με τη λογική των επιμέρους συστατικών, τα συστατικά αυτά δεν είναι πάντα άμεσα επαναχρησιμοποιήσιμα. Προκειμένου να προσαρμοστούν στην κάθε νέα εφαρμογή χρειάζεται τροποποίηση του κώδικά τους.

Επιπλέον, σε πολλές περιπτώσεις γίνεται συνδυασμός δύο ή παραπάνω τεχνολογιών. Ο προγραμματιστής χρειάζεται να αποκτήσει γνώσεις σε διαφορετικές τεχνολογίες, με αποτέλεσμα να αυξάνεται συχνά ο χρόνος ανάπτυξης της εφαρμογής.

Λύση στο παραπάνω πρόβλημα φαίνεται να δίνει η μοντελοκεντρική αρχιτεκτονική. Η ύπαρξη μοντέλων για τα συστατικά επιτρέπει την εύκολη και γρήγορη προσαρμογή τους, και τη χρήση τους σε πολλές διαφορετικές εφαρμογές. Έτσι, για παράδειγμα, ένα μοντέλο που περιγράφει το προφίλ ενός χρήστη μιας εφαρμογής θα μπορέσει να χρησιμοποιηθεί από έναν κινητό πράκτορα για να αποθηκεύσει τα στοιχεία του χρήστη που τον δημιούργησε, σε μια εφαρμογή ιστού για την κατηγοριοποίηση των χρηστών, να δημιουργήσει το κατάλληλο σχήμα βάσης δεδομένων αλλά και σε πολλές άλλες εφαρμογές. Το μόνο που απαιτείται είναι η ύπαρξη των κατάλληλων εργαλείων για τον μετασχηματισμό στο κατάλληλο μοντέλο ειδικά για κάθε πλατφόρμα.

5.2 ΓΛΩΣΣΕΣ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ

Όπως γίνεται εύκολα κατανοητό, όταν χρησιμοποιείται μια τέτοια προσέγγιση είναι ιδιαίτερα σημαντικές η επιλογή της μεθόδου περιγραφής του μοντέλου αλλά και ο τρόπος με τον οποίο πραγματοποιείται ο μετασχηματισμός.

Μία γλώσσα μοντελοποίησης είναι μια τεχνητή γλώσσα η οποία επιτρέπει την περιγραφή πληροφοριών ή γνώσης για ένα σύστημα χρησιμοποιώντας μια δομή η οποία είναι που περιγράφεται από ένα σύνολο κανόνων. Οι κανόνες αυτοί επιτρέπουν τη σωστή ερμηνεία της σημασίας των συστατικών που περιγράφονται στην εκάστοτε δομή.

Μια γλώσσα μοντελοποίησης μπορεί να είναι είτε γραφική είτε σε μορφή κειμένου. Οι γραφικές γλώσσες χρησιμοποιούν διαγράμματα με

- σύμβολα τα οποία έχουν κάποιο όνομα και αναπαριστούν έννοιες,
- γραμμές που συνδέουν τα σύμβολα και αναπαριστούν τις σχέσεις τους,
- και γραφικά σχόλια που αναπαριστούν περιορισμούς.

Οι γλώσσα μοντελοποίησης που περιγράφουν το μοντέλο με κείμενο χρησιμοποιούν λέξεις κλειδιά και παραμέτρους ώστε να προσφέρουν εκφράσεις που μπορούν να ερμηνευτούν από κάποιο υπολογιστικό σύστημα.

Οι γλώσσες αυτές δεν είναι απαραίτητα εκτελέσιμες. Ακόμα και σε αυτές που υπάρχει δυνατότητα εκτέλεσής τους, αυτό δε σημαίνει ότι οι προγραμματιστές δεν είναι πλέον απαραίτητοι. Αντιθέτως, οι εκτελέσιμες γλώσσα μοντελοποίησης έχουν σαν σκοπό να ενισχύσουν την παραγωγικότητα έμπειρων προγραμματιστών ώστε να αντιμετωπίσουν πολυπλοκότερα προβλήματα.

5.2.1 UNIFIED MODELLING LANGUAGE

Η Unified Modelling Language (UML) [79] είναι μια γραφική γλώσσα μοντελοποίησης. Η πρώτη έκδοση της UML έγινε διαθέσιμη το 1997 από το Object Management Group (OMG). Ένας από τους στόχους της UML είναι να παρέχει μια σταθερή και κοινή γλώσσα σχεδιασμού για εφαρμογές λογισμικού. Η UML σύντομα έγινε αποδεκτή και αποτελεί πλέον ένα ευρέως διαδεδομένο standard.

Η UML προσφέρει ένα σύνολο από διαγράμματα τα οποία μπορούν να χρησιμοποιηθούν μέσα σε μια μεθοδολογία ανάπτυξης λογισμικού προκειμένου να γίνει ευκολότερη η κατανόηση της εφαρμογής που αναπτύσσεται.

Τα διαγράμματα που ορίζονται από την UML χωρίζονται σε τρεις κατηγορίες:

Διαγράμματα συμπεριφοράς

- Διαγράμματα περιπτώσεων χρήσης (use case diagrams)
- Διαγράμματα καταστάσεων (state machine diagrams)
- Διαγράμματα δραστηριοτήτων (activity diagrams)
- Διαγράμματα επικοινωνίας (communication diagrams)
- Διαγράμματα ακολουθίας (sequence diagrams)

Δομικά διαγράμματα

- Διαγράμματα κλάσεων (class diagrams)
- Διαγράμματα αντικειμένων (object diagrams)

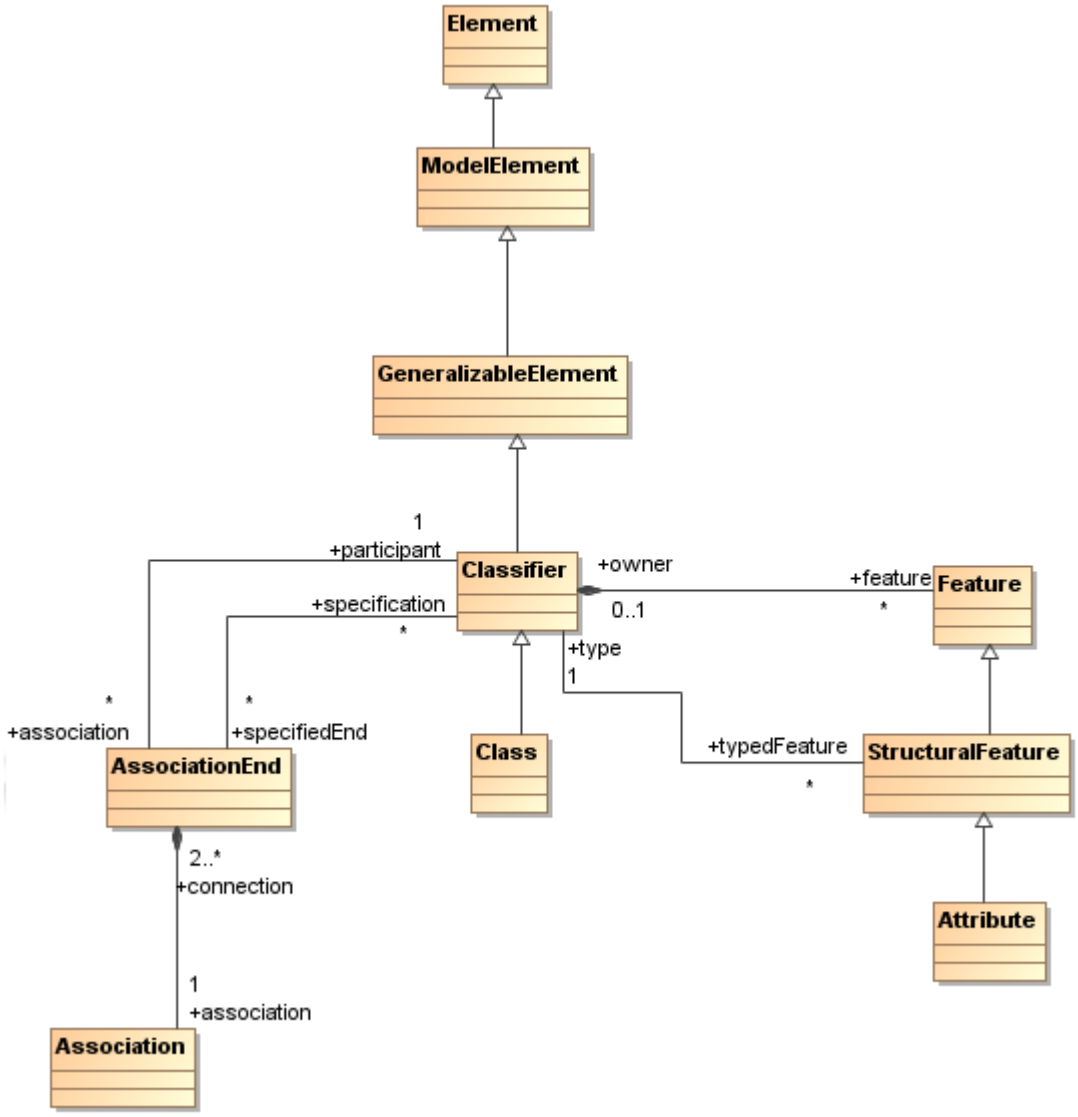
- Διαγράμματα πακέτων (package diagrams)

Αρχιτεκτονικά διαγράμματα

- Διαγράμματα εξαρτημάτων (component diagrams)
- Διαγράμματα ανάπτυξης (deployment diagrams
- Διαγράμματα συστατικών (component diagrams)
- Διαγράμματα συνεργασίας (collaboration diagrams)

Τα διαγράμματα κλάσεων της UML είναι ο πυρήνας της αντικειμενοστραφούς ανάλυσης και σχεδίασης. Τα διαγράμματα αυτά δείχνουν τις κλάσεις του συστήματος, τις σχέσεις τους (συμπεριλαμβανόμενων της κληρονομικότητας – inheritance - , της συνάθροισης – aggregation - και των συσχετίσεων - associations), τα πεδία και τις λειτουργίες των κλάσεων. Τα διαγράμματα κλάσεων χρησιμοποιούνται για πολλούς διαφορετικούς σκοπούς όπως είναι η περιγραφή του μοντέλου ενός προβλήματος, την περιγραφή εννοιών και την λεπτομερή σχεδίαση μοντέλων.

Τα διαγράμματα καταστάσεων αναπαριστούν τις διάφορες καταστάσεις που ένα αντικείμενο μπορεί να πάρει, καθώς και τις μεταβάσεις ανάμεσα σε αυτές τις καταστάσεις. Μια κατάσταση (state) αντιπροσωπεύει ένα στάδιο στη συμπεριφορά του αντικειμένου. Όπως και στα διαγράμματα δραστηριοτήτων της UML είναι δυνατό να υπάρχουν αρχικές και τελικές καταστάσεις. Μια αρχική κατάσταση ομομάζεται και κατάσταση δημιουργίας, και είναι η κατάσταση που έχει το αντικείμενο όταν δημιουργείται, ενώ μια τελική κατάσταση είναι μια κατάστασή απ'όπου δεν υπάρχουν μεταβάσεις εξόδου. Μια μετάβαση είναι η μεταβολή της κατάστασης του αντικειμένου και πραγματοποιείται από κάποιο γεγονός είτε εσωτερικό είτε εξωτερικό ως προς το αντικείμενο.



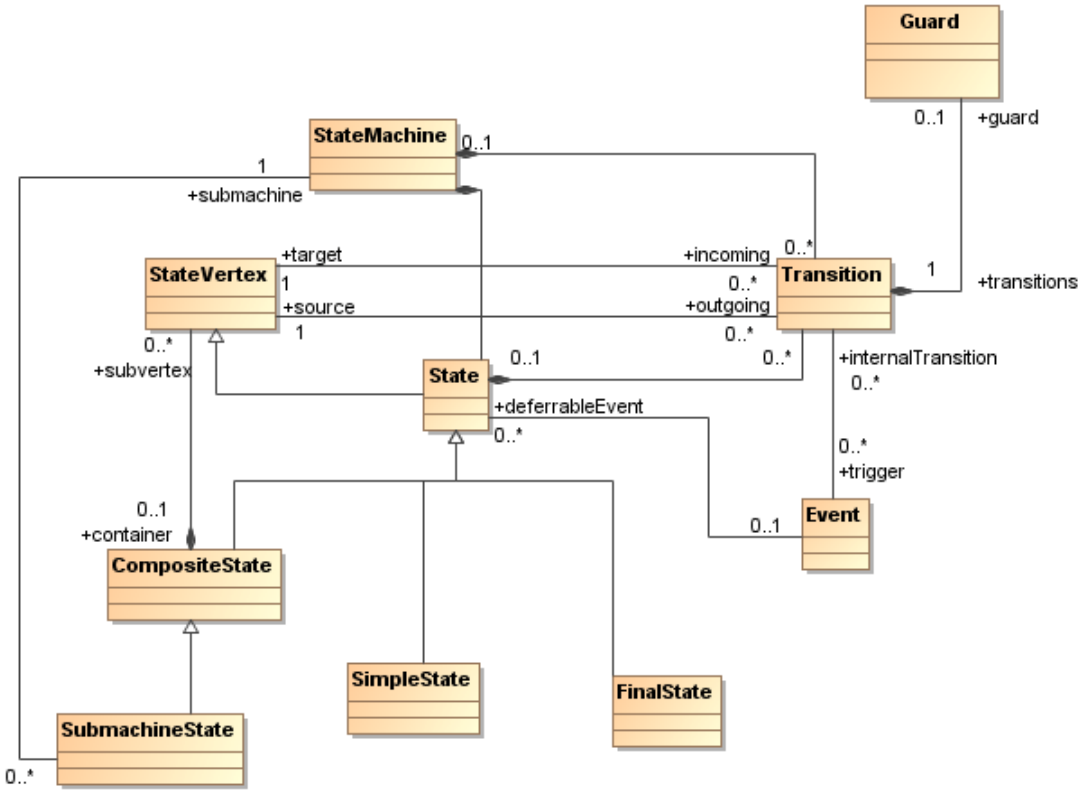
Εικόνα 5.1: Μοντέλο διαγράμματος κλάσεων UML

Η UML προσφέρει τρεις μηχανισμούς επεκτασιμότητας:

1. Tagged values. Μέσω του μηχανισμού αυτού είναι δυνατό να προστεθούν αφηρημένες πληροφορίες στα στοιχεία του μοντέλου. Οι χρήστες μπορούν να προσθέσουν νέες ιδιότητες σε οποιοδήποτε στοιχείο του μοντέλου. Μία τέτοια «ετικέτα» είναι ουσιαστικά ένα ζεύγος λέξης κλειδιού – τιμής που περιγράφει μια συγκεκριμένη ιδιότητα. Κάθε στοιχείο του μοντέλου έχει ένα σύνολο από τέτοια ζεύγη τα οποία κατηγοριοποιούνται βάσει των λέξεων κλειδιών.
2. Stereotypes. Επιτρέπουν την ταξινόμηση των στοιχείων του μοντέλου. Τα stereotypes μπορούν να χρησιμοποιηθούν για να εισάγουν επιπρόσθετες

κατηγοριοποιήσεις ανάμεσα στα στοιχεία του μοντέλου, οι οποίες δεν υποστηρίζονται ρητά από το μεταμοντέλο της UML. «Εφαρμόζοντας» ένα stereotype σε ένα στοιχείο του μοντέλου ουσιαστικά παράγουμε ένα εξειδικευμένο στοιχείο μοντέλου. Αυτός ο μηχανισμός επεκτασιμότητας μπορεί να συγκριθεί με την κληρονομικότητα.

3. Constraints. Επιτρέπουν την εισαγωγή νέων σημασιολογικών περιορισμώ. Με αυτό τον τρόπο είναι δυνατή ο ορισμός επιπρόσθετων περιορισμών που πρέπει να τηρούν τα στοιχεία.



Εικόνα 5.2: Μοντέλο διαγράμματος καταστάσεων UML

5.3 ΜΟΝΤΕΛΟΚΕΝΤΡΙΚΉ ΑΡΧΙΤΕΚΤΟΝΙΚΉ

Η μοντελοκεντρική αρχιτεκτονική (Model Driven Architecture – MDA) [80] είναι μία πρόταση του OMG για τη διαδικασία ανάπτυξης λογισμικού. Η κύρια ιδέα της MDA είναι η αξιοποίηση των μοντέλων στην ανάπτυξη εφαρμογών λογισμικού. Σύμφωνα με την MDA, η διαδικασία ανάπτυξης λογισμικού κατευθύνεται από τη διαδικασία μοντελοποίησης του συστήματος λογισμικού.

Ο κύκλος ανάπτυξης της MDA φαίνεται στην Εικόνα 5.3 και δεν διαφέρει πολύ από τον κλασικό κύκλο ανάπτυξης. Η διαφορά έγκειται στο ότι σε κάθε φάση παράγονται μοντέλα σε μορφή που μπορούν να γίνουν κατανοητά από υπολογιστές.

Ανεξάρτητο από τη πλατφόρμα μοντέλο

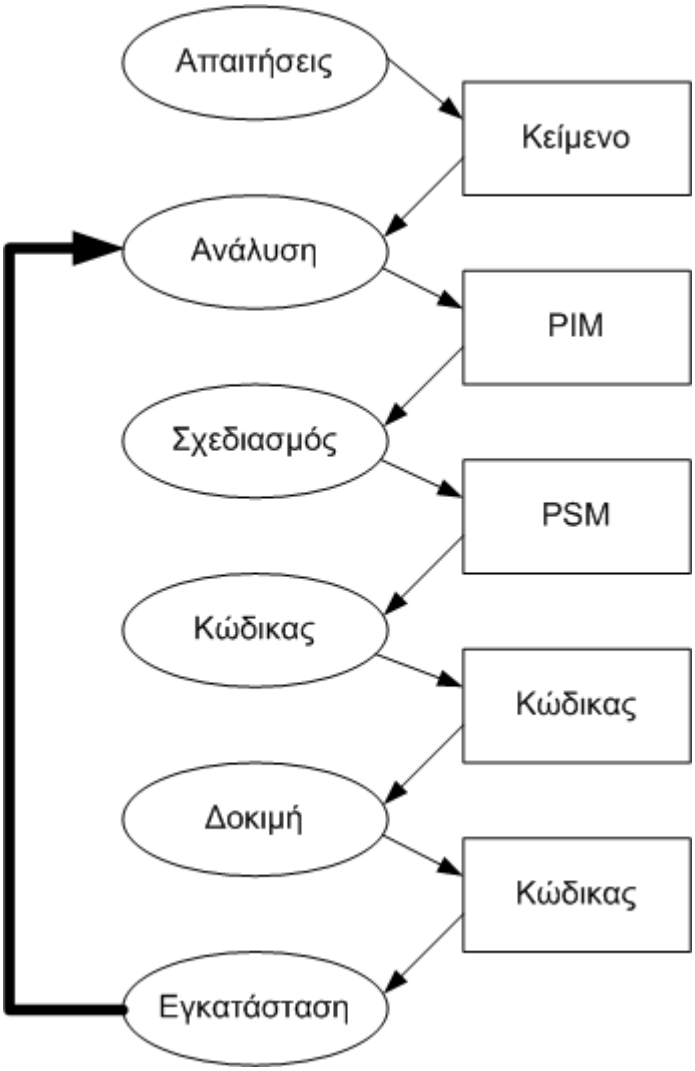
Το ανεξάρτητο από τη πλατφόρμα μοντέλο (Platform Independent Model – PIM) ορίζει ένα μοντέλο υψηλού επιπέδου ανεξάρτητο από τη τεχνολογία στην οποία θα γίνει η υλοποίησή του. Το μοντέλο αυτό περιγράφει το σύστημα λογισμικού που θα αναπτυχθεί με τον τρόπο που κρίνεται καλύτερος για τη συγκεκριμένη εφαρμογή, ανεξάρτητα της τεχνολογίας που τελικά θα χρησιμοποιηθεί.

Μοντέλο ειδικά για τη πλατφόρμα

Στο επόμενο βήμα, το PIM μετατρέπεται σε ένα μοντέλο ειδικά για την πλατφόρμα (Platform Specific Model - PSM). Το PSM ορίζει τις δομές που θα χρησιμοποιηθούν στην πλατφόρμα που θα πραγματοποιηθεί η υλοποίηση. Ένα PIM μπορεί να μετατραπεί σε παραπάνω από ένα PSM, αν χρησιμοποιούνται παραπάνω από μία τεχνολογικές πλατφόρμες για την εφαρμογή.

Κώδικας

Το τελευταίο βήμα κατά τη διαδικασία ανάπτυξης είναι η μετατροπή του PSM σε κώδικα. Μιας και το PSM είναι άμεσα συνδεδεμένο με την τεχνολογία υλοποίησης, ο μετασχηματισμός αυτός είναι άμεσος.



Εικόνα 5.3: Κύκλος ανάπτυξης εφαρμογής λογισμικού με MDA

Από τα παραπάνω γίνεται φανερό ότι η ανάπτυξη μιας εφαρμογής χρησιμοποιώντας την μοντελοκεντρική αρχιτεκτονική προσθέτει επιπλέον αφαιρετικότητα στον τρόπο σχεδιασμού μέσω του PIM. Το πρόβλημα που εμφανίζεται είναι ότι ο μετασχηματισμός από ένα PIM σε ένα PSM και από PSM σε κώδικα είναι χρονοβόρα διαδικασία. Ωστόσο αυτή η διαδικασία μπορεί να αυτοματοποιηθεί π.χ. με εργαλεία που παράγουν κώδικα.

Η εισαγωγή της μοντελοκεντρικής αρχιτεκτονικής στη διαδικασία ανάπτυξης λογισμικού μπορεί να γίνει με πολλούς διαφορετικούς τρόπους και σε πολλά διαφορετικά σημεία. Τα κυριότερα εργαλεία που υπάρχουν και οι δυνατότητες που προσφέρουν συνοψίζονται στον . Εκτός από αυτά τα ολοκληρωμένα εργαλεία, εφαρμογές της MDA υπάρχουν σε πολλές άλλες περιπτώσεις, όπου το μοντέλο υπάρχει σε μορφή XMI και οι μετασχηματισμοί είναι γραμμένοι σε XSL.

Όνομα	**Περιγραφή**	**Open Source**
Kermeta [98]	Το Kermeta βασίζεται στο περιβάλλον του ευρέως διαδεδομένου περιβάλλοντος ανάπτυξης Eclipse. Προσφέρει ένα αντικειμενοστραφές περιβάλλον μετα-προγραμματισμού, όπου είναι δυνατή η χρησιμοποίηση μετα-μοντέλων, ο ορισμός αφαιρετικής σύνταξης, στατικής αλλά και δυναμικής σημασιολογία και η εφαρμογή μετασχηματισμών στο μοντέλο. Έχει αναπτυχθεί σαν μια επέκταση του Eclipse EMF.	ΝΑΙ
MOFScript [99]	Εργαλείο που επιτρέπει τη μετατροπή ενός μοντέλου σε κείμενο, βασισμένο στο στάνταρτ του OMG MOF Model to Text Transformation submissions. Είναι και αυτό βασισμένο στο μεταμοντέλο EMF του Eclipse.	ΝΑΙ
The IBM Model Transformation Framework (MTF) [100]	Διαθέσιμο από το IBM alphaWorks, βασίζεται στο EMF για να προσφέρει δυνατότητες μετασχηματισμού του μοντέλου.	ΝΑΙ
The ATL Engine [101]	Βρίσκεται στον πυρήνα του Eclipse M2M project. Είναι ένα σύνολο από επεκτάσεις του Eclipse, οι οποίες επιτρέπουν την εφαρμογή μετασχηματισμών και την εκτέλεσή τους, ενώ παρέχονται και δυνατότητες αποσφαλμάτωσης. Χρησιμοποιεί μια γλώσσα μετασχηματισμών παρόμοια με τη QVT.	ΝΑΙ
ModFact [102]	Βασίζεται στη γλώσσα TRL και προσφέρει μία μηχανή για αποθήκευση των μετασχηματισμών (MOF Repository).	ΝΑΙ
Kent Modelling Framework (KMF) [103]	Ένα εργαλείο που επιτρέπει τη δημιουργία γλωσσών που υποστηρίζουν το δυναμικό έλεγχο περιορισμών.	ΝΑΙ
OpenArchitectureWare	Ένα ευέλικτο εργαλείο που χρησιμοποιεί	ΝΑΙ

[104]	πρότυπα για τη δημιουργία του τελικού κώδικα και χρησιμοποιεί σαν είσοδο μοντέλα σε μορφή XMI.	
OpenMDX [105]	Περιβάλλον μοντελοκεντρικής αρχιτεκτονικής ανοιχτού κώδικα, που μπορεί να συνεργαστεί με αρκετά άλλα εργαλεία χρησιμοποιώντας το XMI. Υποστηρίζει τη δημιουργία κώδικα για επιλεγμένες πλατφόρμες (J2EE, .Net).	ΝΑΙ
AndroMDA [106]	Το πιο διαδεδομένο εργαλείο, χρησιμοποιεί πρότυπα αρχεία για την παραγωγή κώδικα για την πλατφόρμα J2EE χρησιμοποιώντας σαν είσοδο διαγράμματα UML σε μορφή XMI. Χρησιμοποιεί τη δικιά του γλώσσα προτύπων VTL (Velocity Template Engine) και το API Netbeans MDR για τη διαχείριση των μοντέλων.	ΝΑΙ
XDoclet [107]	Ένας από του πρώτους γεννήτορες κώδικα για την πλατφόρμα J2EE. Αν και δεν ακολουθεί πλήρως τις αρχές της μοντελοκεντρικής αρχιτεκτονικής, μπορεί να συνδυαστεί με άλλα εργαλεία όπως το UMT.	ΝΑΙ
Middlegen [108]	Χρησιμοποιεί μια βάση δεδομένων σαν το μοντέλο και χρησιμοποιεί βιβλιοθήκες και εργαλεία όπως τα JSBC, Velocity, Xdoclet και το Ant για να δημιουργήσει κώδικα.	ΝΑΙ
OOMEGA [109]	Επιτρέπει την ανάπτυξη μοντέλων και μετα-μοντέλων για τον ορισμό γλωσσών και της διαδικασίας δημιουργία κώδικα. Υποστηρίζει διάφορες τεχνολογίες (db4objects, Hibernate, Versant, OOMEGA's MemoryDB, XML και SDF) για να αποθηκεύει τα μεταμοντέλα και τα μοντέλα. Η δημιουργία του κώδικα βασίζεται σε πρότυπα αρχεία που είναι γραμμένα σαν σελίδες JSP.	ΝΑΙ

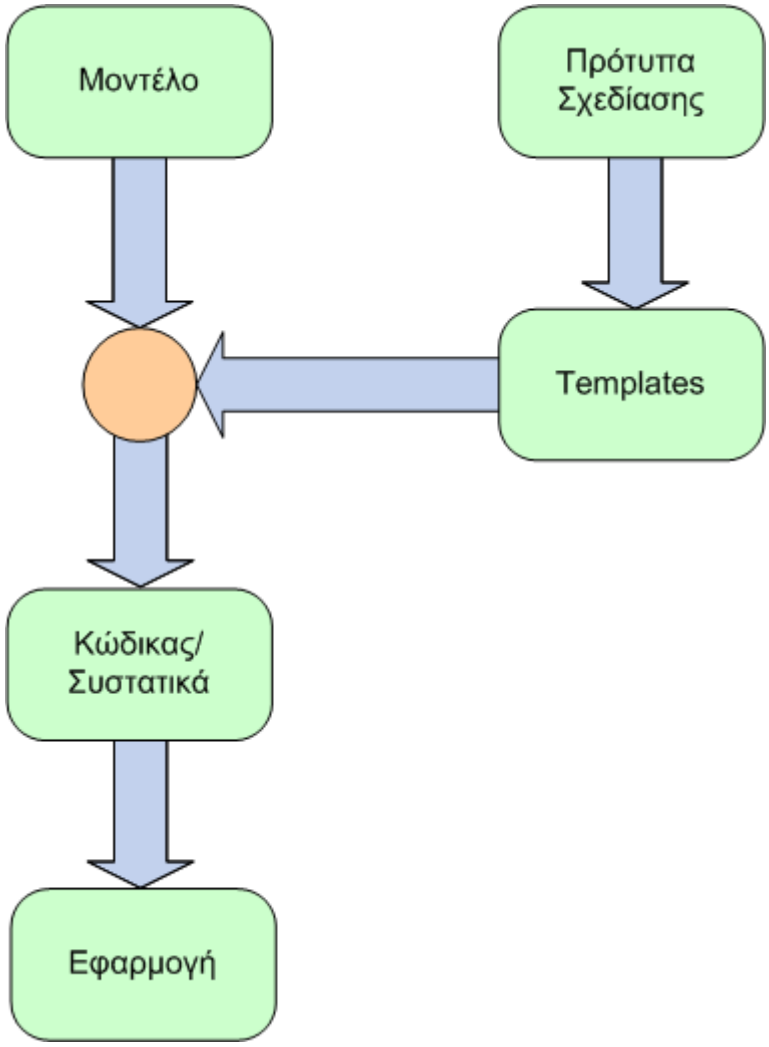
ArcStyler [110]	Πρόκειται για ένα εμπορικό εργαλείο μοντελοκεντρικής αρχιτεκτονικής. Διατίθεται μαζί με το εργαλείο σχεδίασης διαγραμμάτων UML MagicDraw, αλλά υποστηρίζει και άλλα εργαλεία.	ΟΧΙ
MCC (Model Component Compiler) [111]	Υποστηρίζει δημιουργία κώδικα για το περιβάλλον J2EE	ΟΧΙ
OptimalJ [112]	Χρησιμοποιεί notations σε πρότυπα για να επιτύχει μετασχηματισμούς σε PSM. Διαθέτει το δικό του εργαλεία για την ανάλυση της εφαρμογής βάσει σχεδιαγραμμάτων UML.	ΟΧΙ
SosyInc Modeler and Transformation Engine [114]	Προσφέρει μηχανή μετασχηματισμών για τη δημιουργία γραφικών διεπαφών χρήστη (GUI) αλλά και τη δημιουργία κώδικα στη πλευρά του εξυπηρετητή. Χρησιμοποιεί τα μοντέλα OASIS/UML και κανόνες για τον ορισμό της δομής της εφαρμογής.	ΟΧΙ
Model-in-Action and MDA tool suite [115]	Προσφέρει ένα ευέλικτο πλαίσιο για τη δημιουργία κώδικα και μετασχηματισμούς από μοντέλο σε μοντέλο.	ΟΧΙ
MetaEdit [116]	Διαθέτει ενσωματομένο εργαλείο μοντελοποίησης και μετα-μοντελοποίησης ώστε να ορίζονται γλώσσες αλλά καικανόνες για τη δημιουργία κώδικα. ποστηρίζει διεπαφές XML και SOAP/Web Service τόσο για τα μοντέλα όσο και για τα μετα-μοντέλα.	ΟΧΙ
MDWorkbench [117]	Επιτρέπει το μετασχηματισμό κειμένου και μοντέλου μέσω ενός συνόλου από εργαλεία. Υποστηρίζει οποιοδήποτε τύπο μεταμοντέλου σαν είσοδο. Βασίζεται στο περιβάλλον του Eclipse και του EMF.	ΟΧΙ

iQgen 3.0 [118]	Χρησιμοποιεί πρότυπα αρχεία για τη δημιουργία του PSM και του κώδικα τα οποία είναι γραμμένα σαν σελίδες JSP. Μπορεί να χρησιμοποιήσει σαν είσοδο μοντέλα που βρίσκονται σε διαφορετικές μορφές, συμπεριλαμβανομένων των XMI, XML και ECore.	ΟΧΙ

Πίνακας 5.1: Εργαλεία μοντελοκεντρικής αρχιτεκτονικής

5.4 ΜΟΝΤΕΛΟΚΕΝΤΡΙΚΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Όπως φάνηκε στα προηγούμενα κεφάλαια, οι τεχνολογίες κατανεμημένων αντικειμένων προσφέρουν πολλές δυνατότητες και επιτρέπουν σε εφαρμογές να χρησιμοποιήσουν μεγάλο πλήθος πόρων με έναν αυτοματοποιημένο τρόπο. Ωστόσο ένα από τα σημαντικότερα προβλήματα είναι η πολυπλοκότητα της ανάπτυξης μιας τέτοιας εφαρμογής. Λύση σε αυτό το πρόβλημα μπορεί να δώσει η μοντελοκεντρική αρχιτεκτονική.



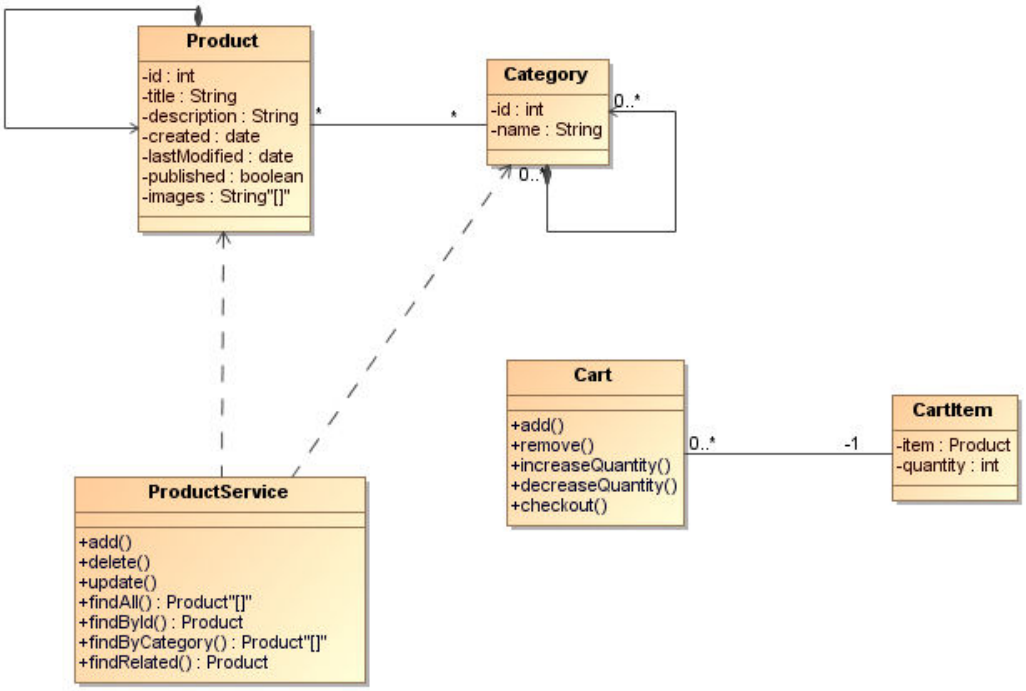
Εικόνα 5.4: Ανάπτυξη εφαρμογής με χρήση μοντελοκεντρικής αρχιτεκτονικής

Η κεντρική ιδέα της προτεινόμενης λύσης αποτελείται από δύο στάδια. Το πρώτο στάδιο είναι η ανάλυση και ο σχεδιασμός των μοντέλων του εκάστοτε προβλήματος με τη χρήση της UML. Χρησιμοποιήθηκε ένα υποσύνολο της γλώσσας UML, και πιο συγκεκριμένα τα διαγράμματα κλάσεων για την περιγραφή του domain model και των λειτουργιών που υπάρχουν διαθέσιμες σε κάθε εφαρμογή, τα διαγράμματα καταστάσεων για την περιγραφή της ροής της εφαρμογής και τα stereotypes έτσι ώστε να οριστούν κάποιοι επιπλέον μηχανισμοί επεκτασιμότητας.

Το δεύτερο στάδιο είναι η αξιοποίηση του μοντέλου και η χρήση προτύπων σχεδίασης για το μετασχηματισμό του μοντέλου σε κώδικα. Τα πρότυπα σχεδίασης μοντελοποιούν συνήθη προβλήματα. Προκειμένου να μπορέσουν να χρησιμοποιηθούν μετατρέπονται σε templates. Τα templates είναι ο τρόπος με τον οποίο μετατρέπουμε το PIM μας σε PSM. Περιέχουν όσες πληροφορίες απαιτείται προκειμένου να μπορέσει να δημιουργηθεί ο κατάλληλος κώδικας. Για κάθε τεχνολογία που χρησιμοποιείται αρκεί να υπάρχουν τα

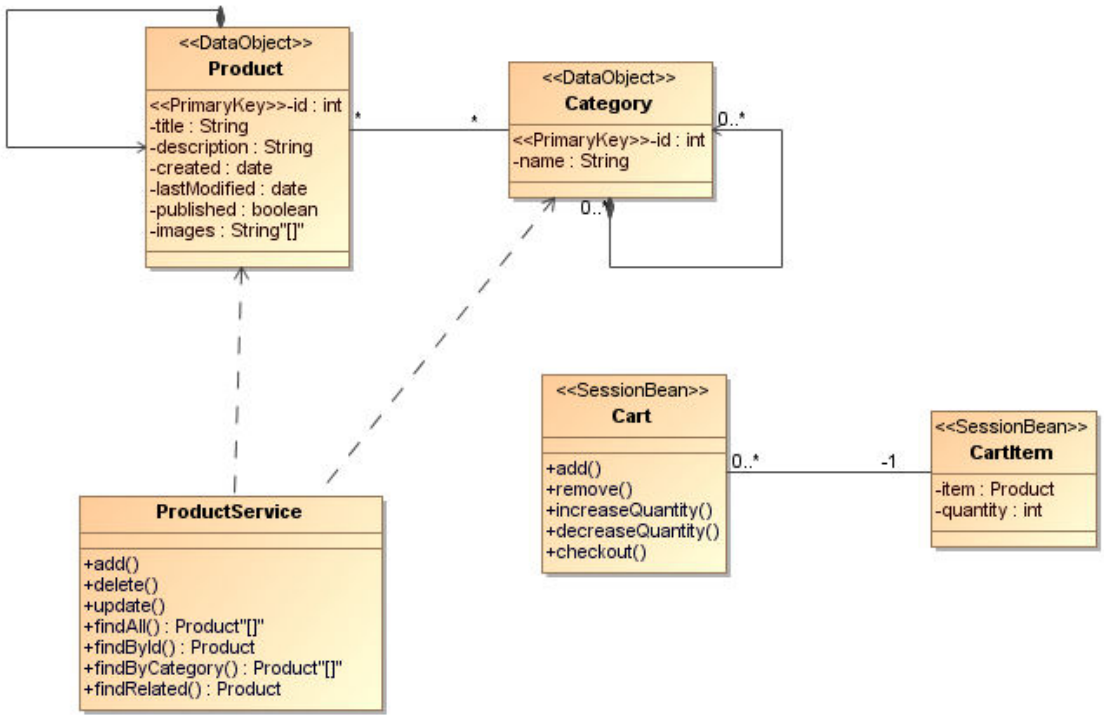
αντίστοιχα templates. Στις περιπτώσεις που απαιτείται η χρήση παραπάνω από μια τεχνολογίας αρκεί να χρησιμοποιηθούν όλα τα κατάλληλα templates.

Η αξία της παραπάνω προσέγγισης μπορεί να γίνει φανερή με ένα απλό παράδειγμα. Έστω μια απλή εφαρμογή ιστού για ένα κατάστημα όπου ο χρήστης μπορεί να δει ένα σύνολο από προϊόντα και να προσθέτει κάποια από αυτά σε ένα καλάθι αγορών. Είτε αν ο προγραμματιστής χρησιμοποιήσει γράψει απευθείας τον κώδικα είτε χρησιμοποιήσει κάποια εργαλεία για να σχεδιάσει το μοντέλο πρώτα, το μοντέλο θα είναι παρόμοιο με αυτό στην Εικόνα 5.5.



Εικόνα 5.5: Παράδειγμα μοντέλου καταστήματος

Το μοντέλο αυτό αναπαριστά τα βασικά αντικείμενα που αποτελούν το μοντέλο της εφαρμογής, αλλά δεν περιέχει καμία πληροφορία για το πώς αυτά υλοποιούνται στο σύστημα. Για παράδειγμα, το καλάθι αγορών και τα αντικείμενα που περιέχει μπορεί να είναι Java Beans τα οποία αποθηκεύονται στη συνεδρία του χρήστη. Αυτή η πληροφορία δεν γίνεται φανερή με κάποιον τρόπο. Σε αυτό ακριβώς μπορούν να βοηθήσουν οι μηχανισμοί επεκτασιμότητας που προσφέρει η UML. Έτσι για παράδειγμα το stereotype <<SessionBean>> μπορούν να εφαρμοστούν στα αντικείμενα που αποθηκεύονται στη συνεδρία, το <<DataObject>> στα αντικείμενα που αποθηκεύονται στη βάση δεδομένων κλπ



Εικόνα 5.6: Παράδειγμα μοντέλου καταστήματος με χρήση stereotypes

Το πρόβλημα γίνεται φανερό σε περιπτώσεις όπου πρέπει να γίνουν αλλαγές στο σύστημα. Στην περίπτωση μιας απλής αλλαγής, για παράδειγμα με την προσθήκη ενός ακόμα πεδίου στην κλάση που αναπαριστά τα προϊόντα, θα πρέπει να γίνουν αλλαγές και στα αρχεία που περιέχουν τις κλάσεις που αποθηκεύουν τα προϊόντα από τη βάση δεδομένων, στις φόρμες που χρησιμοποιεί ο χρήστης κ.ο.κ.. Σε περίπτωση μιας μεγαλύτερης αλλαγής, όπως π.χ. η αλλαγή του εξυπηρετητή που θα εκτελείται η εφαρμογή, θα πρέπει να αλλάξουν πληροφορίες σε πολλά αρχεία που περιέχουν metadata για την εφαρμογή και να χαθεί αρκετός χρόνος.

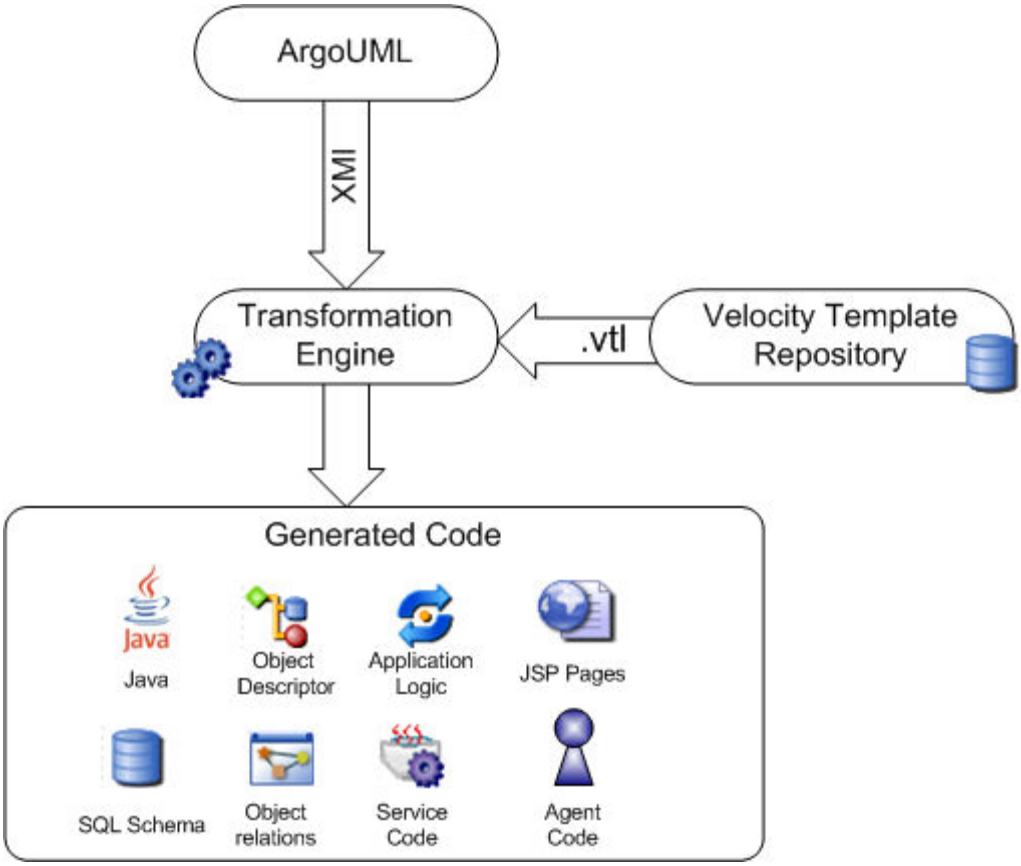
Στο μοντέλο που υπάρχει στην Εικόνα 5.6 περιέχονται όλες οι πληροφορίες που αντιστοιχούν τη λογική του μοντέλου με τη φυσική αναπαράστασή του. Έτσι σε περίπτωση που αλλάξει η υποδομή αρκεί να χρησιμοποιηθεί ο κατάλληλος μετασχηματισμός για τη δημιουργία του τελικού κώδικα. Με άλλα λόγια, αυτό είναι το PIM μοντέλο μας. Οι μετασχηματισμοί που θα του εφαρμόσουμε θα οδηγήσουν στο PSM μοντέλο.

Αυτή είναι και η βασική ιδέα της εφαρμογής. Σημασιολογικές πληροφορίες για τα αντικείμενα και τις έννοιες που αποτελούν το μοντέλο της εφαρμογής δηλώνονται ως stereotypes. Το κάθε stereotype αντιστοιχίζεται σε έναν ή παραπάνω μετασχηματισμούς. Ο κάθε μετασχηματισμός είναι αυτός που είναι υπεύθυνος να συλλέξει τα στοιχεία του

μοντέλου που θα χρησιμοποιήσει σαν είσοδο στη διαδικασία του μετασχηματισμού. Με αυτόν τον τρόπο είναι δυνατή η κατηγοριοποίηση των αντικειμένων και η παραγωγή του επιπλέον κώδικα που απαιτείται.

5.5 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΡΓΑΛΕΙΟΥ ΜΟΝΤΕΛΟΚΕΝΤΡΙΚΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

Η αρχιτεκτονική του εργαλείου μοντελοκεντρικής αρχιτεκτονικής παρουσιάζεται συνοπτικά στην Εικόνα 5.7.



Εικόνα 5.7: Γενική αρχιτεκτονική εργαλείου μοντελοκεντρικής αρχιτεκτονικής

Ο πυρήνας της εφαρμογής είναι η μηχανή μετασχηματισμών. Το συστατικό αυτό παίρνει σαν είσοδο το μοντέλο της εφαρμογής και ένα σύνολο από πρότυπα αρχεία και δημιουργεί τον τελικό κώδικα. Τα δεδομένα του μοντέλου της εφαρμογής περιγράφονται σε ένα αρχείο σε μορφή XML Metadata Interchange (XMI) [92]. Το αρχείο αυτό περιέχει:

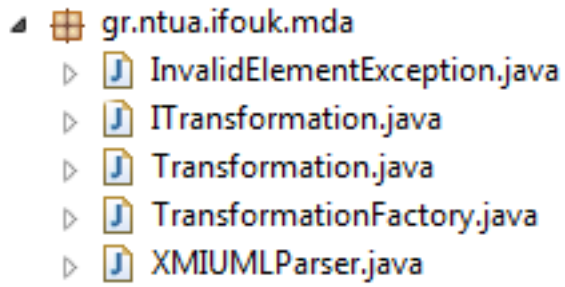
- Τις κλάσεις που συνθέτουν το πρόβλημα και αναπαριστούν το μοντέλο τις εφαρμογής
- Διαγράμματα καταστάσεων που αναπαριστούν τη ροή της εφαρμογής.

Ο κύριος λόγος της χρησιμοποίησης του XMI είναι ότι αποτελεί ευρέως διαδεδομένο standard, με αποτέλεσμα πολλά εργαλεία UML να προσφέρουν δυνατότητες εξαγωγής των μοντέλων τους σε μορφή XMI. Παραδείγματα τέτοιων εργαλείων είναι το ArgoUML [96]

και το MagicDraw [97]. Στα στοιχεία που περιέχονται στα διαγράμματα UML έχουν εφαρμοστεί stereotypes. Το κάθε stereotype μπορεί να χρησιμοποιείται από έναν η περισσότερους μετασχηματισμούς. Η ανάγνωση αυτών των αρχείων από τη μηχανή μετασχηματισμών γίνεται με χρήση των βιβλιοθηκών JMI [94] και MDR [93].

Εκτός από το μοντέλο της εφαρμογής, απαραίτητη είναι και η ύπαρξη κάποιων αρχείων που περιέχουν τα πρότυπα για τη δημιουργία του κώδικα. Τα αρχεία αυτά είναι γραμμένα χρησιμοποιώντας τη σύνταξη της Velocity Template Language. Η γλώσσα αυτή είναι κομμάτι της βιβλιοθήκης Velocity [95], η οποία επιτρέπει την δημιουργία templates τα οποία χρησιμοποιούν κλάσεις Java για να συμπληρώσουν τις παραμέτρους. Παρόλο που είναι απλή είναι αρκετά ισχυρή, και υποστηρίζει όλες τις δομές ελέγχου που υπάρχουν στις παραδοσιακές γλώσσες προγραμματισμού. Σε κάθε μετασχηματισμό μπορούν να αντιστοιχούν παραπάνω από ένα πρότυπα αρχεία.

Στη συνέχεια παρουσιάζονται οι βασικές κλάσεις της εφαρμογής.

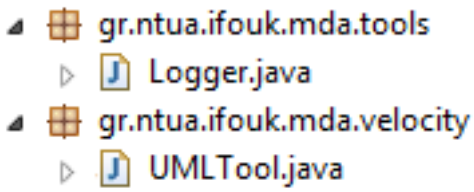


Διεπαφή ITransformation: Η διεπαφή ITransformation ορίζει τις βασικές λειτουργίες που πρέπει να προσφέρει ένας μετασχηματισμός. Πιο συγκεκριμένα, θα πρέπει να ελέγχει αν σε ένα αντικείμενο έχουν εφαρμοστεί τα απαραίτητα stereotypes ώστε να συμμετάσχει στη διαδικασία του μετασχηματισμού και να πραγματοποιεί το μετασχηματισμό χρησιμοποιώντας το κατάλληλο template.

Abstract Κλάση Transformation: Η κλάση αυτή προσφέρει βασικές βοηθητικές λειτουργίες που είναι χρήσιμες σε ένας μετασχηματισμό. Τέτοιες λειτουργίες είναι η ύπαρξη και διαχείριση μιας λίστας αναφορών στα αντικείμενα που θα συμμετάσχουν στο μετασχηματισμό, και η ύπαρξη μιας λίστας με τα stereotypes που θα έχουν σημασία για το μετασχηματισμό.

Κλάση TrasfomationFactory: Ο ρόλος της κλάσης αυτής είναι να διαχειρίζεται τους διαθέσιμους μετασχηματισμούς. Είναι ένα Singleton το οποίο εσωτερικά αποθηκεύει όλους τους διαθέσιμους μετασχηματισμούς και επιτρέπει σε εξωτερικά συστατικά να τους διαχειρίζονται.

Κλάση XMIUMLParser: Μία από τις σημαντικότερες κλάσεις τις εφαρμογής. Αναλαμβάνει να διαβάσει ένα αρχείο εισόδου σε μορφή XMI και να ενημερώσει τους κατάλληλους μετασχηματισμούς. Χρησιμοποιεί τις βιβλιοθήκες JMI και Netbeans MDR για την ανάγνωση του αρχείου και την αναπαράστασή του εσωτερικά.



Κλάση Logger: Η κλάση αυτή προσφέρει γενικές λειτουργίες καταγραφής μηνυμάτων κατά το χρόνο εκτέλεσης της εφαρμογής.

Κλάση UMLTool: Βοηθητική κλάση που χρησιμοποιείται μέσα στα templates. Κύριος λόγος ύπαρξης είναι να παρέχει μεθόδους που εκτελούν εργασίες οι οποίες εμφανίζονται συχνά μέσα σε ένα template. Παράδειγμα τέτοιων μεθόδων είναι η αντιστοίχηση τύπων δεδομένων ανάμεσα σε διαφορετικές γλώσσες, η διαχείριση των ιδιοτήτων μιας κλάσης, η εύρεση του τύπου μιας κλάσης κ.ο.κ.

5.6 Σχεδιαστικα Προτυπα

Το πρώτο βήμα για την ανάπτυξη των εργαλείων που διευκολύνουν την παραπάνω διαδικασία ήταν η ανάπτυξη των κατάλληλων μετασχηματισμών για συνήθη σχεδιαστικά πρότυπα (design patterns). Ένα σχεδιαστικό πρότυπο είναι μια λύση που χρησιμοποιείται συχνά για να λύσει προβλήματα που εμφανίζονται συχνά στη διαδικασία ανάπτυξης λογισμικού.

Η πρώτη αναφορά στα σχεδιαστικά πρότυπα γίνεται στο [81]. Το βιβλίο αυτό αν και απευθύνεται στην αρχιτεκτονική, περιέχει ιδέες που μπορούν να εφαρμοστούν σε πολλά άλλα πεδία, συμπεριλαμβανομένου και αυτό της σχεδίασης λογισμικού. Μάλιστα η πρώτη εφαρμογή των ιδεών που περιγράφονται στο παραπάνω βιβλίο ήρθε το 1987. Στο [84] περιγράφονται τα πρώτα σχεδιαστικά πρότυπα για το σχεδιασμό διεπαφών χρήστη, ενώ λίγο αργότερα δημοσιεύτηκαν τα πρώτα πρότυπα για τη γλώσσα προγραμματισμού C++. Το πιο σημαντικό βιβλίο για τα σχεδιαστικά πρότυπα είναι το [85], όπου υπάρχει εκτενής αναφορά στα πιο χαρακτηριστικά και συχνά χρησιμοποιούμενα σχεδιαστικά πρότυπα. Τα σχεδιαστικά πρότυπα χρησιμοποιούνται κατά κόρον για το σχεδιασμό των σύγχρονων συστημάτων λογισμικού [84], [85].

Τα σχεδιαστικά πρότυπα μπορούν να χωριστούν σε τρεις διαφορετικές κατηγορίες:

1. Creational. Πρόκειται για τα σχεδιαστικά πρότυπα που ασχολούνται με διάφορους μηχανισμούς δημιουργίας αντικειμένων. Σε πολλές περιπτώσεις ο κλασικός τρόπος δημιουργίας αντικειμένων δεν προσφέρει τον απαιτούμενο έλεγχο πάνω στη διαδικασία της δημιουργίας, οπότε είναι απαραίτητη η εισαγωγή μηχανισμών ελέγχου της διαδικασίας.

2. Structural. Τα σχεδιαστικά πρότυπα αυτής της κατηγορίας διευκολύνουν το σχεδιασμό ενός συστήματος προσφέροντας έναν εύκολο τρόπο για τον ορισμό των σχέσεων ανάμεσα σε άλλα αντικείμενα.
3. Behavioral. Είναι σχεδιαστικά πρότυπα που εντοπίζουν και υλοποιούν κάποιους πρότυπους μηχανισμούς επικοινωνίας ανάμεσα σε άλλα αντικείμενα.

Στη συνέχεια παρουσιάζονται τα πιο διαδεδομένα σχεδιαστικά πρότυπα (
Πίνακας 5.2).

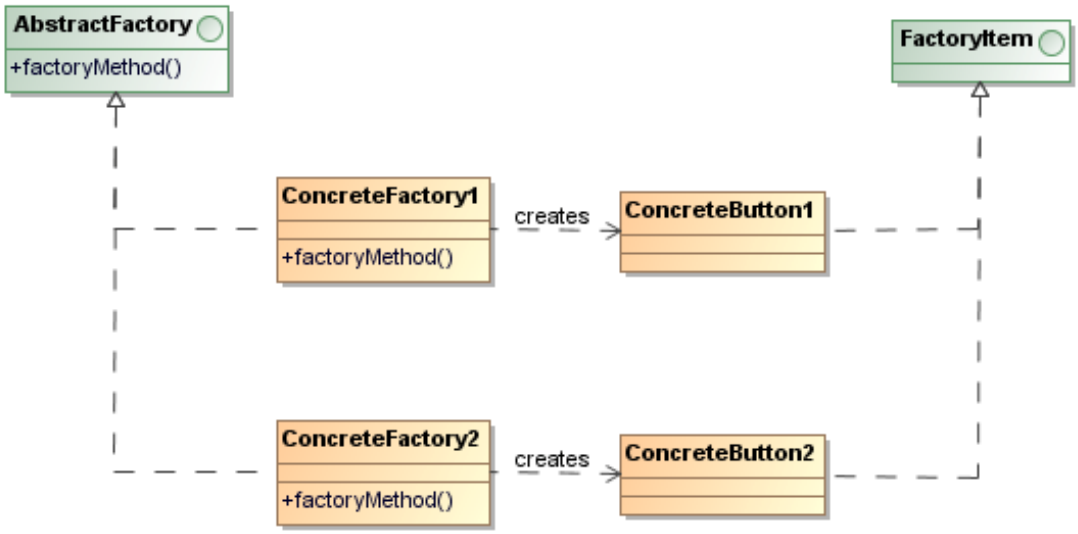
Creational		
Όνομα	**Περιγραφή**	**Παράρτημα**
Abstract factory	Προσφέρει μία διεπαφή για την δημιουργία αντικειμένων που ανήκουν σε μία «οικογένεια» αντικειμένων, χωρίς να ορίζεται η κλάση.	5.6.1
Object pool	Επιτρέπει την επαναχρησιμοποίηση αντικειμένων που έχουν ιδιαίτερες απαιτήσεις σε πόρους, τοποθετώντας ένα περιορισμένο αριθμό από αυτά σε ένα σύνολο.	0
Singleton	Επιβάλει τη δημιουργία ενός μοναδικού αντικειμένου μιας τάξης και προσφέρει μία διεπαφή για την πρόσβαση σε αυτό.	5.6.3
Structural		
Όνομα	**Περιγραφή**	**Παράρτημα**
Adapter	Μετατρέπει τη διεπαφή μιας τάξης σε μια άλλη διεπαφή. Με αυτό τον τρόπο επιτρέπει κλάσεις που αλλιώς δεν θα μπορούσαν να συνεργαστούν λόγο της διαφορετικής διεπαφής να γίνουν συμβατές.	5.6.5
Proxy	Προσφέρει ένα ενδιάμεσο αντικείμενο το οποίο χειρίζεται την πρόσβαση σε ένα άλλο αντικείμενο.	5.6.4

Behavioral		
Όνομα	**Περιγραφή**	**Παράρτημα**
Command	Ενθυλακώνει μια αίτηση σαν ένα αντικείμενο επιτρέποντας με αυτόν τον τρόπο την παραμετροποίηση των αιτήσεων.	
Iterator	Προσφέρει έναν τρόπο πρόσβασης στα στοιχεία ενός σύνθετου αντικειμένου σειριακά χωρίς όμως να αποκαλύπτεται η υποδομή που αποθηκεύονται τα δεδομένα.	5.6.6
Observer	Επιτρέπει σε αντικείμενα να ειδοποιούνται για αλλαγές στην κατάσταση κάποιων άλλων αντικειμένων.	5.6.8

Πίνακας 5.2: Διαδεδομένα σχεδιαστικά πρότυπα

5.6.1 ABSTRACT FACTORY

Το σχεδιαστικό πρότυπο «Abstract Factory» (Εικόνα 5.8) προσφέρει μία μέθοδο ώστε να ενθυλακώνονται ανεξάρτητα εργοστάσια αντικειμένων που ακολουθούν κάποιες γενικές αρχές. Η πρόσβαση σε αυτά τα εργοστάσια γίνεται μέσω μίας κοινής διεπαφής. Το αντικείμενο που χρησιμοποιεί αυτά τα εργοστάσια δεν γνωρίζει (ή δεν χρειάζεται να γνωρίζει) ποιο από τα «συμπαγή» εργοστάσια χρησιμοποιεί, παρά μόνο τη διεπαφή που τα ορίζει. Τα αντικείμενα που παράγονται και αυτά υλοποιούν μία συγκεκριμένη διεπαφή. Με αυτόν τον τρόπο οι λεπτομέρειες της υλοποίησης αποσυνδέονται από τον τρόπο χρήσης των αντικειμένων. Χαρακτηριστικό παράδειγμα χρησιμοποίησης του πρότυπου αυτού είναι στο σχεδιασμό γραφικών διεπαφών ανεξάρτητων από το λειτουργικό σύστημα.



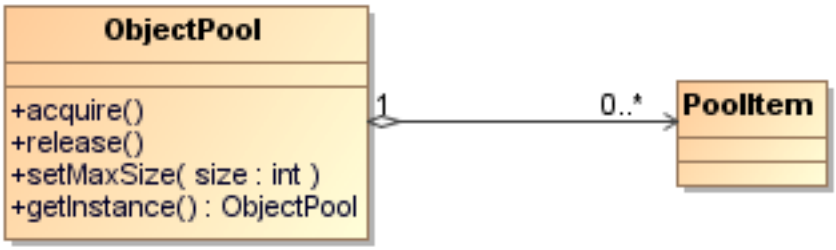
Εικόνα 5.8: Σχεδιαστικό πρότυπο Abstract Factory

Οι οντότητες που πρέπει να αναγνωρίζει η μηχανή μετασχηματισμών είναι οι κλάσεις που αναπαριστούν τα συμπαγή εργοστάσια και τις μεθόδους που δημιουργούν τα αντικείμενα. Αυτό γίνεται εύκολα εφαρμόζοντας το stererotype <<FactoryMethod>> στη μέθοδο της κλάσης. Η διεπαφή δημιουργείται αυτόματα.

5.6.2 OBJECT POOL

Ένα Object Pool είναι ένα σύνολο από αντικείμενα που έχουν ήδη αρχικοποιηθεί και φυλάσσονται προς χρήση. Συνήθως το σύνολο αυτό αποτελείται από αντικείμενα που απαιτούν πολλούς πόρους ή αρκετό χρόνο για να αρχικοποιηθούν. Μία κλάση πελάτης αντί να δημιουργήσει ένα τέτοιο αντικείμενο απευθείας, ζητάει από το Object Pool μια αναφορά σε ένα από αυτά που έχουν δημιουργηθεί, και αν υπάρχει κάποιο διαθέσιμο τη λαμβάνει. Όταν ολοκληρώσει την εργασία της με αυτό το αντικείμενο το απελευθερώνει.

Για την περίπτωσή μας αντικείμενα οποιαδήποτε κλάσης μπορούν να δηλωθούν ως αντικείμενα που ανήκουν σε ένα Object Pool αρκεί να τους εφαρμοστεί το stereotype <<Poolable>>.

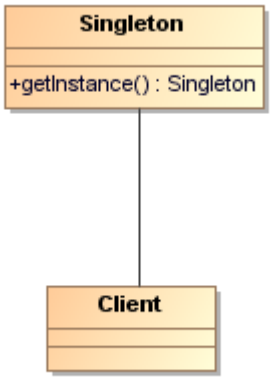


σ

Εικόνα 5.9: Σχεδιαστικό πρότυπο Object Pool

5.6.3 SINGLETON

Το σχεδιαστικό πρότυπο Singleton επιβάλει τη δημιουργία ενός μοναδικού αντικειμένου μιας τάξης και προσφέρει μία διεπαφή για την πρόσβαση σε αυτό. Η λύση που ακολουθείται είναι ο κατασκευαστής της κλάσης να γίνεται ιδιωτικός (private) και να δημιουργείται ένα αντικείμενο της κλάσης είτε όταν φορτώνεται η κλάση είτε την πρώτη φορά που γίνεται χρήση της διεπαφής που επιστρέφει μια αναφορά σε αυτό το αντικείμενο. Η διεπαφή είναι μια στατική (static) μέθοδος, η οποία συνήθως έχει το όνομα getInstance().



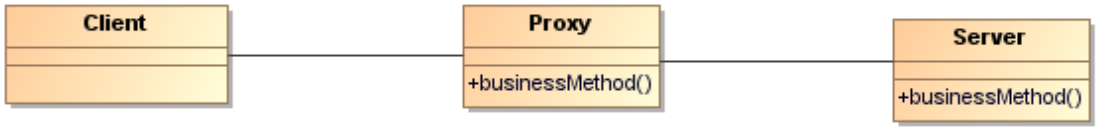
Εικόνα 5.10: Σχεδιαστικό πρότυπο Singleton

Στην περίπτωσή μας το template που χρησιμοποιήθηκε είναι αυτό που παρουσιάζεται στο Παράρτημα 8.2.4. Προκειμένου να οριστεί ένα αντικείμενο ως Singleton αρκεί να του εφαρμοστεί το sterotype <<Singleton>>.

5.6.4 PROXY

Το σχεδιαστικό πρότυπο Proxy χρησιμοποιείτε ώστε να υπάρχει ένα ενδιάμεσο αντικείμενο ανάμεσα στο αντικείμενο που επεξεργάζεται τις αιτήσεις και στο αντικείμενο το οποίο πραγματοποιεί την κλήση. Η χρησιμότητα του γίνεται φανερή σε περιπτώσεις που θέλουμε

να ελέγχουμε την πρόσβαση σε κάποιο αντικείμενο, χωρίς όμως να αλλάξουμε τη λογική του αντικείμενου.



Εικόνα 5.11: Σχεδιαστικό πρότυπο Proxy

Τα templates που αναπτύχθηκαν είναι δύο. Το ένα αφορά τη δημιουργία Proxy για κλήση μεθόδων τοπικών αντικειμένων. Προκειμένου να δημιουργηθεί ο κώδικας του Proxy αρκεί να εφαρμόσουμε το stereotype <<Proxy>> στο αντικείμενο. Το δεύτερο επιτρέπει τη δημιουργία Proxy αντικειμένων που λειτουργούν ως Web Services.

5.6.5 ADAPTER

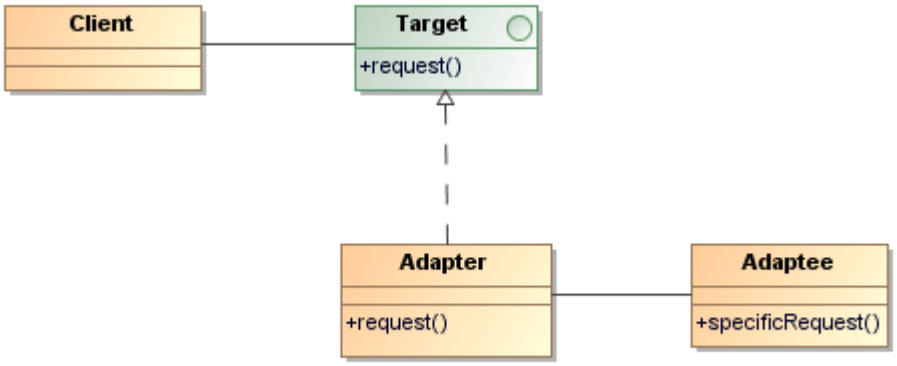
Το σχεδιαστικό πρότυπο Adapter επιτρέπει τη μετατροπή μιας διεπαφής που δεν μπορεί να χρησιμοποιηθεί αλλιώς. Η χρησιμότητα αυτού του προτύπου γίνεται φανερή σε περιπτώσεις που:

- Είναι αναγκαία η επαναχρησιμοποίηση προϋπάρχοντος κώδικα,
- χρειάζεται η δημιουργία μιας κλάσης που συνεργάζεται με άλλες κλάσεις οι οποίες δεν μπορούν να προβλεφθούν εκ των προτέρων και
- σε περιπτώσεις που είναι επιθυμητή η χρησιμοποίηση κάποιων υποκλάσεων αλλά δεν είναι πρακτική η αλλαγή των διεπαφών όλων αυτών των υποκλάσεων.

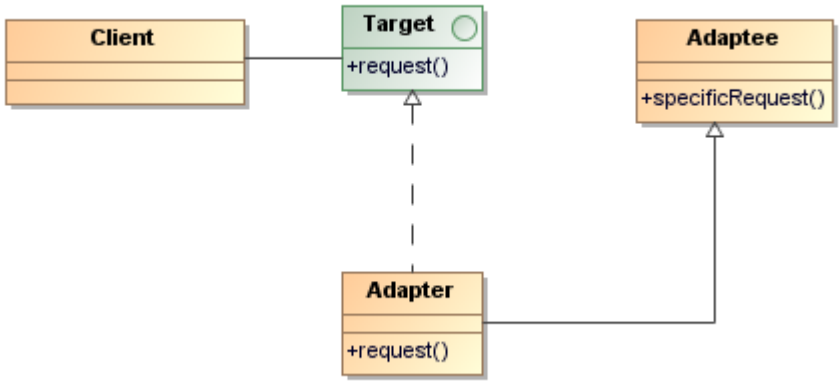
Η υλοποίηση του σχεδιαστικό πρότυπο αυτού μπορεί να γίνει με δύο διαφορετικούς τρόπους, με συσχέτιση (Εικόνα 5.12) και με κληρονομικότητα (Εικόνα 5.13). Η διαφορά ανάμεσα σε αυτές τις δύο προσεγγίσεις είναι ότι στη χρησιμοποίηση Adapter με συσχέτιση, η μετατροπή δουλεύει για ένα συγκεκριμένο αντικείμενο, ενώ με τη χρησιμοποίηση της κληρονομικότητας ο Adapter λειτουργεί για την κλάση και όλες τις υποκλάσεις της.

Η δημιουργία του κώδικα γίνεται εύκολα, αρκεί να έχουν εφαρμοστεί δύο stereotypes:

- <<Adapter>>, το οποίο εφαρμόζεται στη μέθοδο του Adapter και
- <<Adaptee>>, που εφαρμόζεται στη μέθοδο της κλάσης Adaptee



Εικόνα 5.12: Σχεδιαστικό πρότυπο Adapter με συσχέτιση

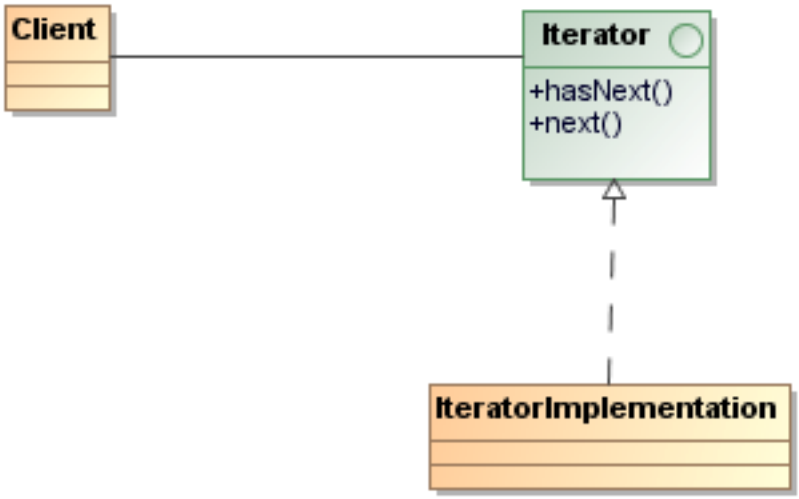


Εικόνα 5.13: Σχεδιαστικό πρότυπο Adapter με κληρονομικότητα

5.6.6 ITERATOR

Το σχεδιαστικό πρότυπο Iterator χρησιμοποιείτε σε περιπτώσεις που είναι επιθυμητή η ύπαρξη μιας λίστας από παρόμοια αντικείμενα χωρίς να χρειάζονται να εμφανιστούν η λεπτομέρειες της εσωτερικής αναπαράστασης της λίστας. Οι λειτουργίες που προσφέρει αυτή η λίστα είναι δύο: αν υπάρχει άλλο αντικείμενο μετά την τρέχουσα θέση στη λίστα, και η εύρεση του επόμενου αντικειμένου.

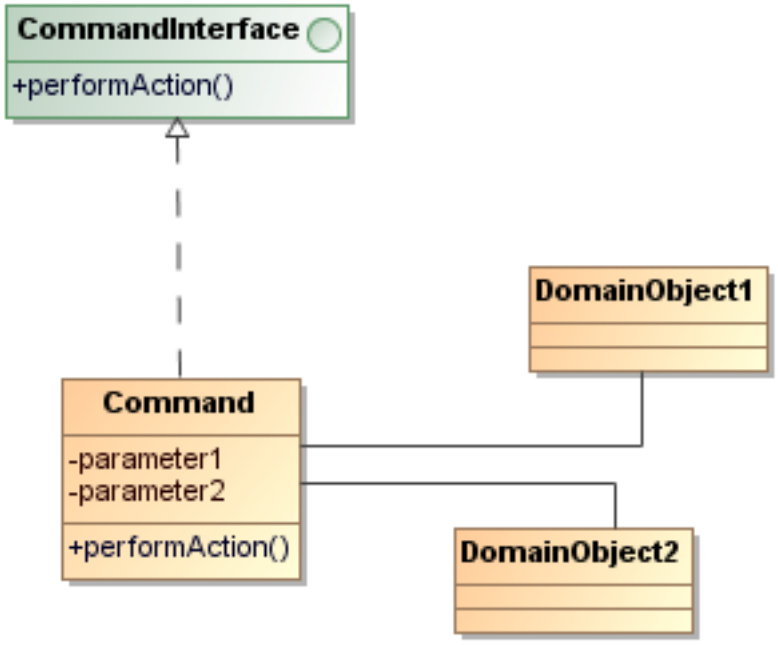
Για την προτεινόμενη πλατφόρμα έχουν υλοποιηθεί δύο διαφορετικά είδη Iterators. Το πρώτο, που χρησιμοποιεί το stereotype <<Iterator>>, επιτρέπει την διάταξη αντικειμένων μια κλάσης σε μια γραμμική λίστα. Το δεύτερο, <<TreeIterator>> επιτρέπει τη διαχείριση των αντικειμένων που βρίσκονται σε δομή δέντρου.



Εικόνα 5.14: Σχεδιαστικό πρότυπο Iterator

5.6.7 COMMAND

Το Command είναι ένα σχεδιαστικό πρότυπο το οποίο επιτρέπει την ενθυλάκωση μιας συγκεκριμένης εντολής σε μια κλάση. Παρόλο που μια εντολή μπορεί να αναπαρασταθεί ως μια μέθοδος ενός αντικειμένου, υπάρχουν αρκετοί λόγοι που κάνουν το συγκεκριμένο σχεδιαστικό πρότυπο χρήσιμο. Έτσι, η κλάση της εντολής μπορεί να χρησιμοποιηθεί σαν το ενδιάμεσο προσωρινό χώρο που αποθηκεύονται οι παράμετροι της εντολής, καθώς και μετα-δεδομένα όπως ποιος εκτέλεσε την εντολή και πότε. Επιπλέον, με αυτό τον τρόπο οι εντολές μπορούν να εισαχθούν σε δομές δεδομένων όπως πχ λίστες οι οποίες κρατάνε την προτεραιότητά τους ή στοίβες έτσι ώστε να υποστηρίζονται λειτουργίες αναίρεσης.



Εικόνα 5.15: Σχεδιαστικό πρότυπο Command

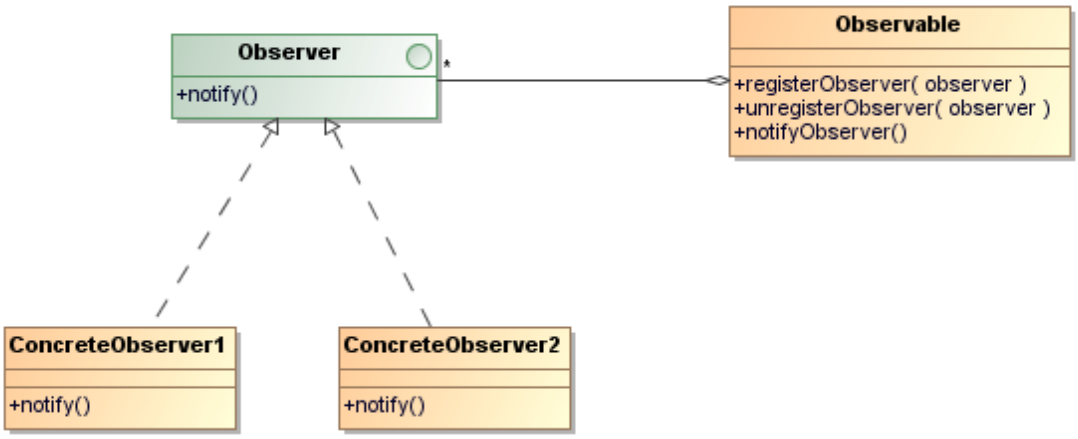
Στην περίπτωσή μας, το σχεδιαστικό πρότυπο αυτό ενθυλακώνει κλήσεις σε κάποια μέθοδο ενός αντικειμένου. Για όσες μεθόδους εφαρμοστεί το stereotype <<Command>> δημιουργείτε και η αντίστοιχη κλάση Command.

5.6.8 OBSERVER

Το σχεδιαστικό πρότυπο Observer (γνωστό και ως publish/subscribe) χρησιμοποιείτε για την επίβλεψη της κατάστασης ενός αντικειμένου. Οι κλάσεις που θέλουν να ειδοποιηθούν για την αλλαγή υλοποιούν μία συγκεκριμένη διεπαφή (Observer), η οποία ορίζει τη μέθοδο η οποία θα καλείτε όταν αλλάζει η κατάσταση του αντικειμένου. Τα αντικείμενα των οποίων η κατάσταση επιβλέπεται από τα άλλα πρέπει να έχουν τρεις βασικές μεθόδους:

1. registerObserver – η οποία προσθέτει έναν Observer σε μια λίστα με τα αντικείμενα που επιβλέπουν το τρέχον αντικείμενο,
2. unregisterObserver – η οποία αφαιρεί έναν Observer από την παραπάνω λίστα, και
3. notifyObserver – η οποία αναλαμβάνει να ειδοποιήσει όλους τους Observers που βρίσκονται στη λίστα για τις αλλαγές.

Ο μηχανισμός αυτός συναντάται πολύ συχνά κατά το σχεδιασμό γραφικών διεπαφών χρήστη, αλλά και σε πολλά συστήματα υπηρεσιών ιστού, όπου οι εφαρμογές – πελάτες πρέπει να ειδοποιούνται για αλλαγές σε κάποιες παραμέτρους του συστήματος.
Εφαρμόζοντας σε μια κλάση το stereotype <<Observable>>, τότε δημιουργείτε ο αντίστοιχος κώδικας για την διαχείριση των κλάσεων που θα επιβλέπουν το αντικείμενο, καθώς και η διεπαφή που αυτές πρέπει να επεκτείνουν.



Εικόνα 5.16: Σχεδιαστικό πρότυπο Observer

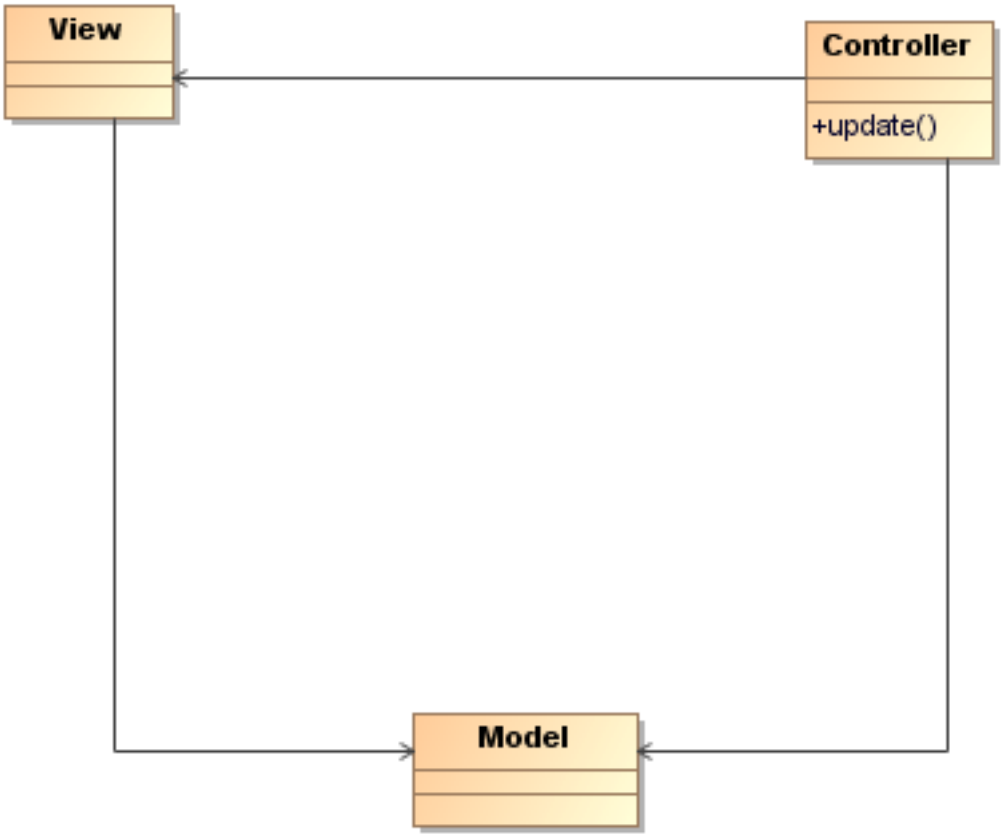
5.7 MODEL - VIEW - CONTROLLER

Μία από τις πιο συχνά εμφανιζόμενες μορφές προγραμμάτων είναι αυτά που ανάλογα με τις εντολές του χρήστη δημιουργούν, ενημερώνουν ή διαγράφουν δεδομένα που υπάρχουν σε κάποιο αρχείο ή βάση δεδομένων. Οι εφαρμογές αυτές εμφανίζονται σε πολλά διαφορετικά περιβάλλοντα, από εφαρμογές ιστού μέχρι εφαρμογές στη γραμμή εντολών. Το μοντέλο που χρησιμοποιείτε συνήθως για την ανάπτυξη τέτοιων εφαρμογών είναι το Model-View-Controller (MVC). Η λογική αυτού του προτύπου σχεδίασης είναι ότι το πρόβλημα χωρίζεται σε τρία κύρια συστατικά:

- Το μοντέλο του προβλήματος (model) που διαχειρίζεται τη συμπεριφορά και τα δεδομένα στο πεδίο της εφαρμογής. Το μοντέλο επιστρέφει πληροφορίες σχετικά με την κατάστασή του στις όψεις και αντιδρά σε αιτήσεις για αλλαγή στην κατάστασή του που προέρχονται από τον ελεγκτή.
- Οι όψεις (views), οι οποίες είναι ο τρόπος με τον οποίο αναπαριστώνται τα δεδομένα. Αξίζει να σημειώσουμε ότι οι όψεις δεν είναι αναγκαίο να είναι μια διεπαφή προς τον χρήστη, αλλά μπορεί να είναι μια διεπαφή για άλλα εξωτερικά συστατικά.

- Ο ελεγκτής (controller) είναι το συστατικό το οποίο ενθυλακώνει τη λογική της εφαρμογής και αναλαμβάνει το συντονισμό της. Επεξεργάζεται τις αιτήσεις που προέρχονται από τις όψεις και αφού εκτελέσει τις απαραίτητες ενέργειες στέλνει αιτήσεις προς το μοντέλο για να πραγματοποιήσει αλλαγές στην κατάστασή του.

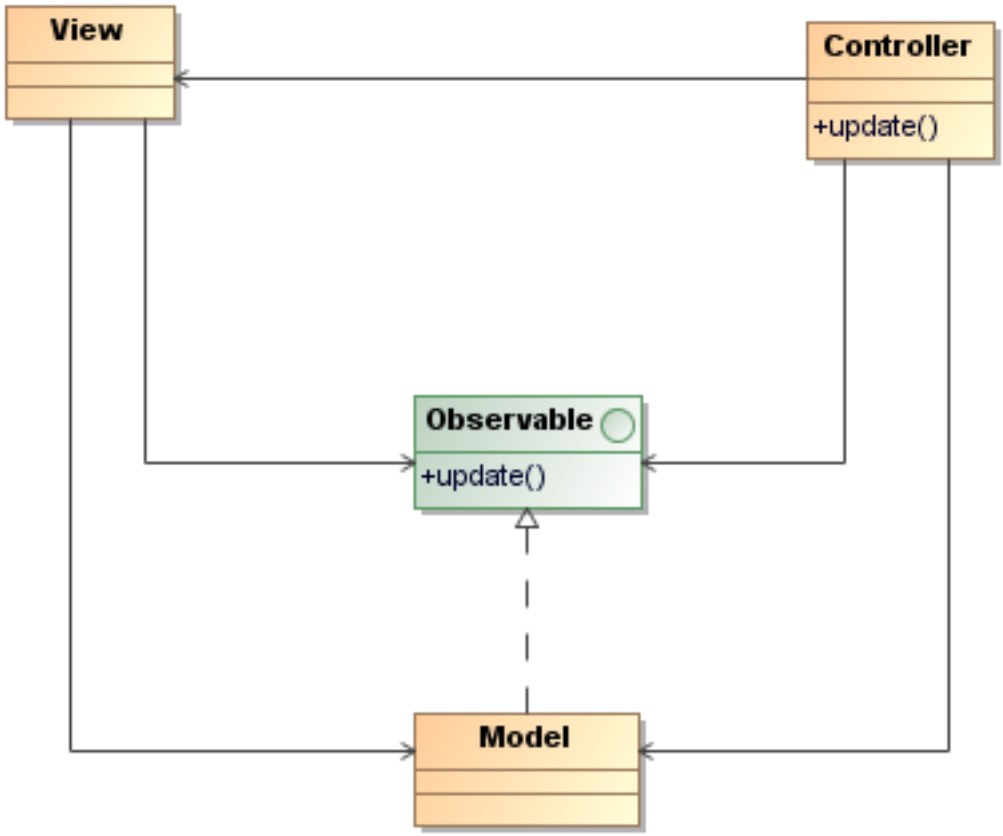
Πολλές φορές ο ελεγκτής και οι όψεις χρησιμοποιούν το σχεδιαστικό πρότυπο Observer για να παρακολουθούν αλλαγές στο μοντέλο. Οι δύο αυτές προσεγγίσεις φαίνονται στις ακόλουθες εικόνες.



Εικόνα 5.17: Model-View-Controller

Οι βασικότερες λειτουργίες που χρειάζονται για το μεγαλύτερο μέρος των εφαρμογών που αναπτύσσονται με αυτό το σχεδιαστικό πρότυπο είναι η δημιουργία, ανάκτηση, ενημέρωση και διαγραφή των δεδομένων από κάποια πηγή δεδομένων (data source) η οποία μπορεί να είναι ένα σύστημα αρχείων, μια τοπική ή απομακρυσμένη βάση δεδομένων κλπ. Οι λειτουργίες αυτές είναι γνωστές και σαν CRUD από τα αρχικά των λειτουργιών Create-Retrieve-Update-Delete. Έχουν εμφανιστεί αρκετές διαφορετικές προσεγγίσεις που επιτρέπουν την εύκολη δημιουργία των αντικειμένων του μοντέλου μιας εφαρμογής, με πιο εύχρηστη αυτή του Ruby On Rails. Το RoR επιτρέπει στον προγραμματιστή να

δημιουργήσει όλες τις κλάσεις του μοντέλου του και κάποιες όψεις που επιτρέπουν τις λειτουργίες CRUD με λίγες εντολές στη γραμμή εντολών και χρησιμοποιώντας σαν είσοδο τη βάση δεδομένων. Στην περίπτωσή μας οι λειτουργίες αυτές χρησιμοποιούνται για δικτυακές εφαρμογές που αναπτύσσονται με τη γλώσσα προγραμματισμού Java.



Εικόνα 5.18: Model-View-Controler με χρήση του σχεδιαστικού προτύπου Observer

Πιο αναλυτικά, οι απαιτήσεις που υπάρχουν για τις λειτουργίες μας είναι οι εξής:

- Δημιουργία κώδικα κλάσεων των αντικειμένων
- Δημιουργία αντικειμένου σε μια βάση δεδομένων
- Ανάκτηση του αντικειμένου βάσει διαφορετικών κριτηρίων
- Ενημέρωση ήδη υπάρχοντος αντικειμένου
- Διαγραφή αντικειμένου
- Συσχετίσεις ανάμεσα στα αντικείμενα
- Δημιουργία απλών JSP σελίδων που θα επιτρέπουν σε ένα χρήστη τις παραπάνω λειτουργίες

Το μοντέλο μπορεί εύκολα να περιγραφεί χρησιμοποιώντας POJOs. Κατά τη διάρκεια του σχεδιασμού της εφαρμογής αρκεί να εφαρμόσουμε στα αντικείμενα του μοντέλου μας το sterotype <<POJO>>. Το πρότυπο που χρησιμοποιείτε για τη δημιουργία των κλάσεων κατά τον μετασχηματισμό φαίνεται στο Παράρτημα 8.2.1. Για τα αντικείμενα του μοντέλου μας πρέπει να υπάρχουν και οι αντίστοιχες λειτουργίες CRUD. Προκειμένου να πετύχουμε τη χρήση διαφορετικών βάσεων δεδομένων χρησιμοποιήσαμε τη βιβλιοθήκη ανοιχτού κώδικα Hibernate 3. Η βιβλιοθήκη αυτή είναι ευρέως διαδεδομένη και χρησιμοποιείται από πολλές εφαρμογές ανοιχτού κώδικα. Προσφέρει διεπαφές ανεξάρτητες από το επίπεδο αποθήκευσης δεδομένων. Κατά τη διάρκεια του μετασχηματισμού δημιουργείτε μία κλάση για κάθε αντικείμενο του μοντέλου που επιτρέπει να πραγματοποιηθούν οι λειτουργίες CRUD καθώς και τα αντίστοιχα αρχεία ρυθμίσεων (Παράρτημα 8.2.2, 8.2.3).

Οι όψεις αποτελούνται από σελίδες JSP. Μια πρότυπη μορφή τέτοιων σελίδων δημιουργείτε αυτόματα κατά το μετασχηματισμό (Παράρτημα 8.2.2).

Σαν controller χρησιμοποιείτε ένα Java Servlet το οποίο αναλαμβάνει να δέχεται αιτήσεις από τις σελίδες JSP και χρησιμοποιώντας τις κλάσεις που προσφέρουν τις υπηρεσίες να πραγματοποιεί τις αντίστοιχες αλλαγές στο μοντέλο.

5.8 ΕΦΑΡΜΟΓΗ ΣΕ ΚΙΝΗΤΟΥΣ ΠΡΑΚΤΟΡΕΣ

Στο κεφάλαιο 3 περιγράψαμε τις γενικές αρχές και συστατικά μιας πλατφόρμας κινητών πρακτόρων. Ένα ενδιαφέρον ζήτημα στο οποίο μπορεί να βοηθήσει η μοντελοκεντρική αρχιτεκτονική είναι η σχεδίαση της λογικής των κινητών πρακτόρων που εκτελούνται σε αυτή την πλατφόρμα. Όπως αναφέραμε παραπάνω, ο κάθε κινητός πράκτορας είναι μια αυτόνομη οντότητα λογισμικού. Η λογική του κάθε πράκτορα μπορεί να αναπαρασταθεί ως ένα διάγραμμα καταστάσεων. Μάλιστα, στην πράξη, οι ενέργειες που θα εκτελέσει ο κάθε πράκτορας κατά το χρόνο εκτέλεσής του εξαρτώνται από κάποιες μεταβλητές κατάστασης.

Οι πόροι που χρησιμοποιεί ο κάθε πράκτορας κατά τη διάρκεια της εκτέλεσής του μπορούν να χωριστούν στις παρακάτω κατηγορίες:

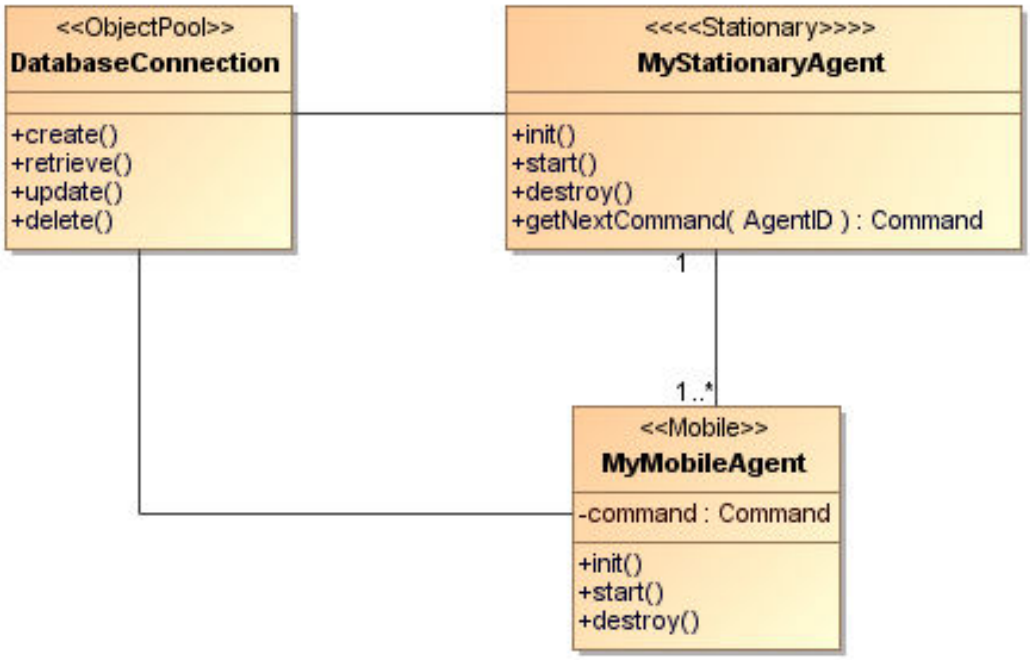
- Αντικείμενα που σχετίζονται με την κατάσταση του πράκτορα. Τα αντικείμενα έχουν διάρκεια ζωής ίδια με αυτή του πράκτορα. Σε περιπτώσεις που ο πράκτορας είναι κινητός, τότε αυτά μεταφέρονται μαζί του όταν αυτός μεταναστεύει.
- Τοπικοί πόροι. Είναι οι πόροι που χρησιμοποιεί ο πράκτορας σε κάθε υπολογιστικό σύστημα στο οποίο εκτελείτε. Παράδειγμα είναι βάσεις δεδομένων και εξοπλισμός με ειδικές διεπαφές. Ο πράκτορας σπάνια αποκτά απευθείας πρόσβαση σε αυτά τα συστατικά. Συνήθως χρησιμοποιείται κάποιο σχεδιαστικό πρότυπο όπως το

Singleton ή το Object Pool. Οι πόροι αυτοί δεσμεύονται όταν ο πράκτορας ξεκινάει την εκτέλεσή του ή μεταναστεύει στο τοπικό σύστημα, και αποδεσμεύονται όταν ολοκληρώσει την εκτέλεσή του και συνεχίσει σε κάποιο άλλο σύστημα.

Η διαδικασία που προτείνεται για τη σχεδίαση ενός συστήματος κινητών πρακτόρων με μοντελοκεντρική αρχιτεκτονική είναι η ακόλουθη:

1. Ορισμός των κλάσεων των πρακτόρων. Οι κλάσεις στις οποίες εφαρμόζεται το stereotype <<Mobile>> είναι οι κινητοί πράκτορες, ενώ σε αυτές που εφαρμόζεται το stereotype <<Stationary>> τους στατικούς.
2. Δημιουργία διαγράμματος κατάστασης κάθε πράκτορα. Για κάθε πράκτορα πρέπει να δημιουργηθεί ένα διάγραμμα κατάστασης με όνομα ίδιο με το όνομα της κλάσης του πράκτορα.

Στην Εικόνα 5.19 παρουσιάζεται το διάγραμμα κλάσεων για το παράδειγμα του μοντέλου Master-Worker που χρησιμοποιήθηκε για την εκτέλεση παραμετρικών εργασιών με χρήση κινητών πρακτόρων. Στην Εικόνα 5.20 φαίνεται το διάγραμμα κατάστασης του πράκτορα – Master και στην Εικόνα 5.21 το διάγραμμα κατάστασης των πρακτόρων – Workers.



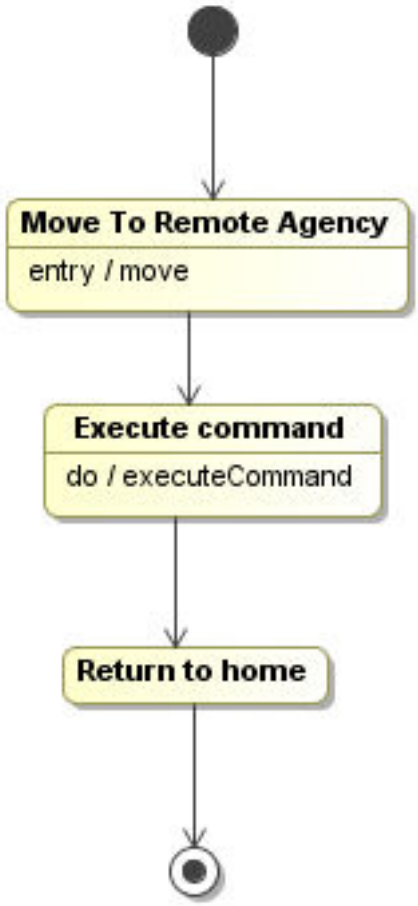
Εικόνα 5.19: Μοντέλο Master-Worker με χρήση τοπικών πόρων από βάση δεδομένων

Εικόνα 5.20: Διάγραμμα κατάστασης πράκτορα Master

Όταν ένα τέτοιο μοντέλο τροφοδοτείτε στην μηχανή μετασχηματισμών, δημιουργείτε ο κατάλληλος κώδικας που περιέχει:

- Κλάσεις πρακτόρων
- Εσωτερική μηχανή καταστάσεων σε κάθε πράκτορα βάσει του διαγράμματος καταστάσεων που του αντιστοιχεί.
- Στις μεθόδους init() (και afterMove() για τους κινητούς πράκτορες) γίνεται η δέσμευση των πόρων για όσα αντικείμενα σχετίζονται με τη κλάση και έχουν κάποιο stereotype από τα <<Poolable>>, <<Singleton>>, <<FactoryMethod>>.
- Στη μέθοδο destroy() (και beforeMove() για τους κινητούς πράκτορες) γίνεται αποδέσμευση των παραπάνω πόρων.

Ο κώδικας των templates που χρησιμοποιούνται υπάρχει διαθέσιμος στο Παράρτημα.



Εικόνα 5.21: Διάγραμμα κατάστασης πράκτορα Worker

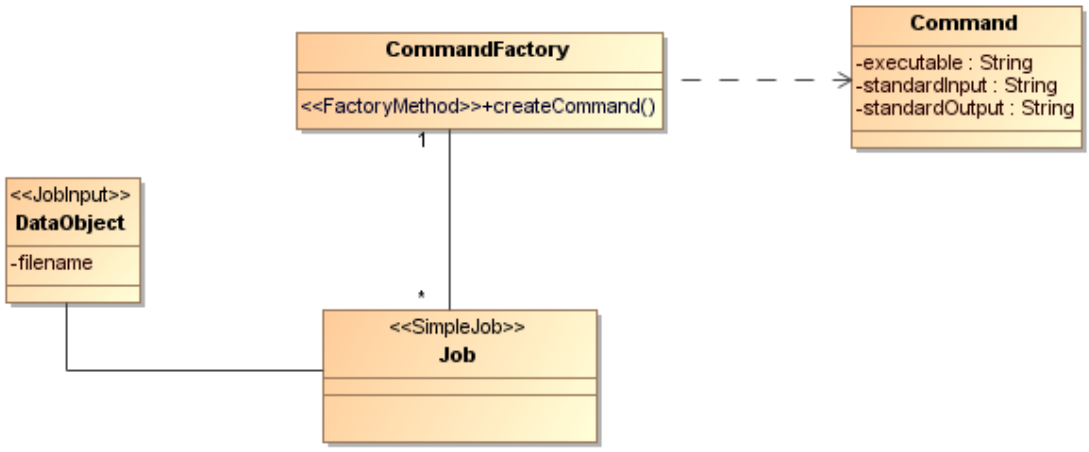
5.9 ΕΦΑΡΜΟΓΗ ΣΤΟ ΠΛΕΓΜΑ

Κατά τη διάρκεια ανάπτυξης της πλεγματικής πύλης έγινε φανερό ότι η αξιοποίηση της τεχνολογίας των Portlet επιτρέπει την επαναχρησιμοποίηση κώδικα για την ανάπτυξη παρόμοιων πλεγματικών πυλών. Τα συστατικά που παραμένουν ίδια σε κάθε περίπτωση είναι αυτά που επιτρέπουν τη διαχείριση των πιστοποιητικών και των εργασιών του χρήστη. Αυτό που αλλάζει είναι η αναπαράσταση των δεδομένων που χρησιμοποιούνται από την εκάστοτε εφαρμογή.

Η λύση που ακολουθήθηκε είναι μια παραλλαγή του Model-View-Controller. Η διαφορά είναι ότι κατά τη διάρκεια του μετασχηματισμού, ο controller δεν υλοποιείτε σαν Java Servlet αλλά σαν ένα Portlet. Έτσι, αυτό που δημιουργείτε τελικά είναι ένα Portlet που επιτρέπει τη διαχείριση των δεδομένων του χρήστη. Ο κώδικας του template για το Portlet υπάρχει στο παράρτημα.

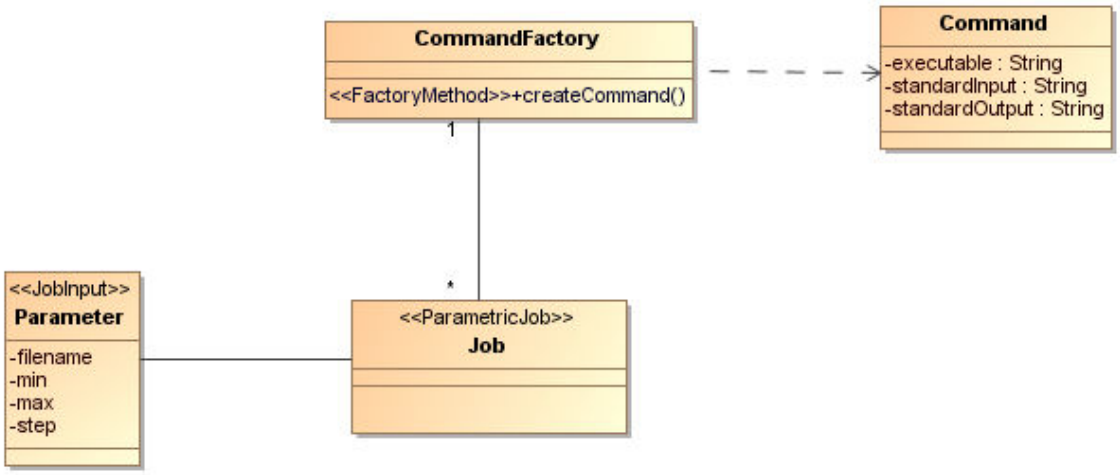
Το δεύτερο σημείο που η μοντελοκεντρική αρχιτεκτονική μπορεί να βοηθήσει είναι στον τρόπο που γίνεται η υποβολή των εργασιών. Οι εργασίες που υποβάλλονται στο Πλέγμα μπορούν να χωριστούν σε τρεις κατηγορίες: τις απλές, τις παραμετρικές και τις εργασίες που αποτελούνται από ένα σύνολο υπο-εργασιών. Οι απλές εργασίες είναι εργασίες που εκτελούνται για μία συγκεκριμένη είσοδο. Οι παραμετρικές επιτρέπουν την εκτέλεση εργασιών για ένα εύρος παραμέτρων. Τέλος, με την υποβολή ανεξάρτητων εργασιών ως ένα σύνολο είναι δυνατή η μείωση του χρόνου που γίνεται η υποβολή των εργασιών. Και για τις τρεις περιπτώσεις χρησιμοποιήσαμε τα διαγράμματα κλάσεων για την δημιουργία των κατάλληλων αρχείων JDL.

Για την περίπτωση της υποβολής απλών εργασιών χρησιμοποιείτε ένα σχήμα σαν αυτό στην Εικόνα 5.22.



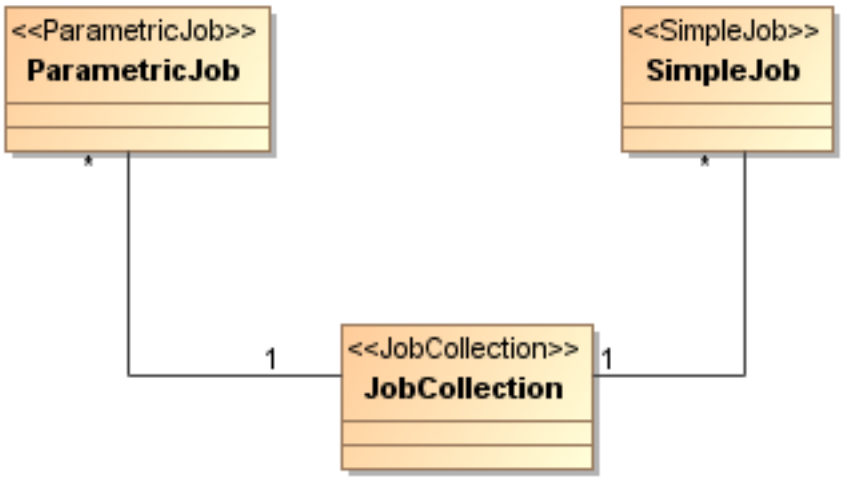
Εικόνα 5.22: Απλές εργασίες για το Πλέγμα

Η κάθε εργασία παίρνει τις πληροφορίες για την εφαρμογή που θα εκτελέσει από ένα «εργοστάσιο» εντολών. Επιπλέον, παίρνει πληροφορίες για την είσοδο από ένα αντικείμενο. Το αντικείμενο αυτό πρέπει να προσφέρει έναν μηχανισμό ώστε να αποθηκεύεται σαν αρχείο. Τα stereotypes που χρησιμοποιούνται είναι το SimpleJob για να ορίζονται οι απλές εργασίες (λειτουργεί ως συνδετικός κρίκος με τα υπόλοιπα συστατικά), το JobInput για την είσοδο της εργασίας, καθώς και το σχεδιαστικό πρότυπο Factory για να συλλέγονται οι πληροφορίες για το πρόγραμμα που θα εκτελεστεί στο Πλέγμα.



Εικόνα 5.23: Παραμετρικές εργασίες

Η λογική για τις παραμετρικές εργασίες είναι παρόμοια, μόνο που σαν είσοδος χρησιμοποιείτε μια κλάση για τον ορισμό των παραμέτρων.



Εικόνα 5.24: Συλλογή εργασιών

Τέλος, για τη συλλογή εργασιών, αρκεί να δημιουργηθεί μία κλάση που έχει εφαρμοστεί το stereotype <<JobCollection>>. Όλες οι παραμετρικές και απλές εργασίες που συνδέονται με συσχέτιση με την κλάση αυτή θεωρούνται ως μέλη του συνόλου και υποβάλλονται στο Πλέγμα σαν ένα ενιαίο σύνολο.

5.10 Πλεονεκτηματα

Τα πλεονεκτήματα της παραπάνω προσέγγισης είναι τα ακόλουθα:

- *Ποιότητα*. Κώδικας που είναι γραμμένος με το χέρι τείνει να έχει περισσότερα σφάλματα και διαφορετική ποιότητα. Ο κύριος λόγος είναι ότι κατά την ανάπτυξή του οι προγραμματιστές βρίσκουν νέες τεχνικές ή βελτιστοποιούν αυτές που χρησιμοποιούν. Μεταφέροντας αυτές τις τεχνικές σε templates είναι ο κώδικας γίνεται ομοιόμορφος, ενώ αποφεύγεται η ύπαρξη σφαλμάτων.
- *Συνέπεια*. Όταν ο κώδικας δημιουργείτε αυτόματα, τότε είναι συνεπής με το σχεδιασμός των διεπαφών, πιο κατανοητός και εύχρηστος.
- *Λιγότερα σημεία παρέμβασης*. Ο σχεδιασμός του λογισμικού παραμένει στο επίπεδο του μοντέλου. Με αυτό τον τρόπο όλες οι αλλαγές συγκεντρώνονται σε ένα συγκεκριμένο σημείο. Για παράδειγμα, αν ο κώδικας γραφόταν με το χέρι, τότε η παραμικρή αλλαγή (π.χ. η αλλαγή του ονόματος σε ένα πίνακα μιας βάσης δεδομένων) θα είχε σαν συνέπεια αλλαγές σε πολλά διαφορετικά σημεία του προγράμματος.
- *Μείωση χρόνου ανάπτυξης του λογισμικού*. Από τη στιγμή που το μεγαλύτερο κομμάτι του κώδικα δημιουργείτε αυτόματα, ο χρόνος που απαιτείτε για την ανάπτυξη του λογισμικού μειώνεται σημαντικά. Αποτέλεσμα είναι ο σχεδιασμός των διεπαφών και των βιβλιοθηκών να γίνεται το σημαντικό κομμάτι κατά την διαδικασία ανάπτυξης του λογισμικού.

6 ΣΥΝΟΨΗ ΚΑΙ ΘΕΜΑΤΑ ΜΕΛΛΟΝΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

6.1 ΣΥΝΟΨΗ ΕΡΕΥΝΗΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Οι τεχνολογίες κατανεμημένων αντικειμένων προσφέρουν πολύτιμα εργαλεία για την επίλυση προβλημάτων με ιδιαίτερες απαιτήσεις σε υπολογιστικούς πόρους. Πολλές διαφορετικές αρχιτεκτονικές και συστήματα έχουν προταθεί ανά καιρούς. Τα συστήματα αυτά προσπαθούν να εκμεταλλευτούν τις εξελίξεις στις τεχνολογίες των δικτύων υπολογιστών αλλά και την πρόοδο στην ισχύ που μπορούν να προσφέρουν οι νέοι υπολογιστές.

Η παρούσα διδακτορική διατριβή αποτέλεσε ένα πολύπλευρο και απαιτητικό έργο με εμπλοκή σε πολλές διαφορετικές τεχνολογίες. Κύριος στόχος ήταν η αξιοποίηση των τεχνολογιών κατανεμημένων αντικειμένων προκειμένου να αναπτυχθούν πλατφόρμες μεσισμικού οι οποίες μπορούν να χρησιμοποιηθούν για μεγάλο πλήθος ετερογενών μεταξύ τους εφαρμογών.

Η πρώτη προσέγγιση που ακολουθήθηκε ήταν η αξιοποίηση των αρχών του κινητού κώδικα μέσω της τεχνολογίας των κινητών πρακτόρων. Προκειμένου να εξασφαλιστεί η ύπαρξη διαθέσιμων υπολογιστικών πόρων μελετήθηκε η ενσωμάτωση μιας πλατφόρμας κινητών πρακτόρων σε εξυπηρετητές ιστού. Επιπλέον, γίνεται δυνατή η αξιοποίηση της υποδομής από εξωτερικές εφαρμογές χρησιμοποιώντας ευρέως διαδεδομένες προδιαγραφές. Για να αξιολογηθεί η ευχρηστία της πλατφόρμας αναπτύχθηκαν δύο δικτυακές εφαρμογές που χρησιμοποιούν κινητούς πράκτορες για να φέρουν εις πέρας τις εργασίες που τους αναθέτει ο χρήστης. Η πρώτη αφορά τη προσομοίωση σύμμορφων στοιχειοκεραιών, ενώ η δεύτερη την ανάπτυξη ενός συστήματος ηλεκτρονικής ψηφοφορίας.

Η τεχνολογία του πλέγματος απασχόλησε ένα μεγάλο μέρος της διδακτορικής διατριβής. Όντας η νεότερη τεχνολογία κατανεμημένων αντικειμένων, επιτρέπει την πρόσβαση σε κατανεμημένους ανά την υφήλιο πόρους μέσω δικτύων υψηλής ταχύτητας. Για να επιτευχθεί η διαλειτουργικότητα ανάμεσα στα επιμέρους συστήματα χρησιμοποιούνται τόσο νέες αλλά και παλιότερες δοκιμασμένες τεχνολογίες σε όλα τα επίπεδα. Μελετήθηκαν οι μηχανισμοί που επιτρέπουν τη διαχείριση και αποστολή εργασιών από το χρήστη στο Πλέγμα. Ωστόσο, όπως και στην περίπτωση των κινητών πρακτόρων, αυτές οι διαδικασίες είναι αρκετά πολύπλοκες για τον απλό χρήστη. Για αυτό το λόγο αναπτύχθηκε δικτυακή πλεγματική πύλη η οποία επιτρέπει στο χρήστη να υποβάλλει και να διαχειριστεί τις εργασίες του με φιλικό προς αυτόν τρόπο.

Η ενασχόληση με αυτές τις δύο τεχνολογίες ανέδειξε ένα σημαντικό πρόβλημα. Παρόλο που όλες οι τεχνολογίες κατανεμημένων αντικειμένων προσφέρουν πρόσβαση σε μεγάλο αριθμό πόρων, ο χρήστης πρέπει διαρκώς να προσαρμόζεται σε νέες και διαφορετικές τεχνολογίες, και σχεδόν πάντα να μετατρέπει τον κώδικά του ώστε να είναι συμβατός με τη νέα τεχνολογία. Προκειμένου να αντιμετωπιστεί το πρόβλημα αυτό μελετήθηκαν οι ιδέες της μοντελοκεντρικής αρχιτεκτονικής. Η προτεινόμενη λύση έχει σαν κεντρική ιδέα την αλλαγή της διαδικασίας ανάπτυξης της κατανεμημένης εφαρμογής. Ο κύριος φόρτος εργασίας μετατοπίζεται στο σχεδιασμό του μοντέλου της εφαρμογής από την συγγραφή του πηγαίου κώδικα. Χρησιμοποιώντας κατάλληλους μετασχηματισμούς, το μοντέλο μετασχηματίζεται σε κώδικα συμβατό με την εκάστοτε πλατφόρμα που θα χρησιμοποιήσει ο χρήστης. Με αυτόν τον τρόπο επιτυγχάνεται εύκολη και γρήγορη επαναχρησιμοποίηση του μοντέλου του χρήστη για πολλά διαφορετικά περιβάλλοντα, μειώνονται τα σφάλματα της εφαρμογής και γίνεται ευκολότερη η δοκιμή των επιμέρους συστατικών της. Το μετα-μοντέλο που χρησιμοποιήθηκε για τον σχεδιασμό των μοντέλων των εφαρμογών είναι αυτό της UML, το οποίο είναι διαδεδομένο και χρησιμοποιείτε για την περιγραφή πολλών διαφορετικών μεταξύ τους συστημάτων.

6.2 ΘΕΜΑΤΑ ΜΕΛΛΟΝΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Η ιδέα της μοντελοκεντρικής αρχιτεκτονικής μπορεί να θεωρηθεί ως επέκταση της παραδοσιακής ιδέας του μεταγλωτιστή έτσι ώστε ένα μοντέλο να μεταγλωτίζεται σταδιακά σε κώδικα. Ενώ όμως η ιδέα αυτή προσφέρει πολλές δυνατότητες, υπάρχουν κάποιες συνέπειες στον τρόπο με τον οποίο γίνεται η διαδικασία της ανάπτυξης:

Ένα πρόβλημα που δεν αντιμετωπίζεται από την μοντελοκεντρική αρχιτεκτονική είναι αυτό της προσθήκης νέων απαιτήσεων. Η προσθήκη μιας νέας απαίτησης έχει σαν αποτέλεσμα αλλαγές στο PIM ή στον τρόπο με τον οποίο γίνονται οι μετασχηματισμοί. Αποτέλεσμα είναι όλη η αλυσίδα των μετασχηματισμών από το PIM μέχρι τον εκτελέσιμο κώδικα να πρέπει να πραγματοποιηθούν ξανά. Σε πολλές περιπτώσεις αυτό μπορεί να είναι ανώδυνο, αλλά σε περιπτώσεις όπως π.χ. οι εφαρμογές ιστού, όπου τέτοιες αλλαγές ισοδυναμούν σε αλλαγές στο περιβάλλον εργασίας του χρήστη και των σελίδων HTML που αυτός χρησιμοποιεί, η διαδικασία εισαγωγής νέων απαιτήσεων είναι χρονοβόρα

Ένα άλλο ζήτημα που πρέπει να αντιμετωπισθεί είναι η συντήρηση της εφαρμογής, η οποία είναι δύσκολη. Ο κύριος λόγος είναι οι πολλαπλοί μετασχηματισμοί που συμβαίνουν. Έτσι, σε περίπτωση που παρατηρηθεί κάποιο σφάλμα, αυτό πρέπει να αναζητηθεί είτε στο PIM είτε σε κάποιον μετασχηματισμό.

Η μονοσήμαντη διαδικασία των μετασχηματισμών κάνει δύσκολη την χρησιμοποίηση κάποιου κώδικα χωρίς να υπάρχει διαθέσιμο το μοντέλο. Παρόλο που γενικά είναι δυνατή η τροποποίηση ενός ενδιάμεσου μοντέλου, μια τέτοια τακτική είναι καλύτερο να μην ακολουθείται, μιας και οδηγεί σε ανακριβή αναπαράσταση του συστήματος. Την επόμενη φορά που θα πραγματοποιηθούν οι μετασχηματισμοί, οι αλλαγές αυτές θα χαθούν.

Ιδιαίτερη σημασία έχει η προσθήκη επιπλέον προτύπων αρχείων για το μετασχηματισμό των μοντέλων στις τελικές εφαρμογές. Για τους κινητούς πράκτορες σημαντικό θα είναι να αναπτυχθούν τα μοντέλα και οι κατάλληλοι μετασχηματισμοί για άλλα μοντέλα κατανομής εργασίας. Διαφορετικές στρατηγικές αποστολής των εργασιών με χρήση των ίδιων πρακτόρων μπορούν να αναπτυχθούν έτσι ώστε ανάλογα τη φύση του προβλήματος να χρησιμοποιείτε η κατάλληλη στρατηγική.

Ένας από τους λόγους για τα παραπάνω προβλήματα είναι η έλλειψη καλά ορισμένων προδιαγραφών για τον τρόπο με τον οποίο πραγματοποιούνται οι μετασχηματισμοί. Ενώ προδιαγραφές όπως η UML και το MOF ορίζουν τις βάσεις για την ανάπτυξη των PIM και PSM, στη διαδικασία των μετασχηματισμών χρησιμοποιούνται πολλές διαφορετικές τεχνολογίες όπως π.χ. XSL, πρότυπα αρχεία ή μετασχηματισμούς γράφων. Η προτεινόμενη λύση βασίζεται σε πρότυπα αρχεία για τη δημιουργία του τελικού κώδικα, μιας και με αυτόν τον τρόπο είναι ευκολότερο να ενσωματωθούν αλλαγές στη σημασιολογία του κώδικα, ωστόσο λύσεις που θα επιτρέπουν τον μετασχηματισμό και από μοντέλο σε μοντέλο πρέπει να διερευνηθούν.

Ένα πεδίο το οποίο έχει ιδιαίτερο ενδιαφέρον είναι αυτό του συνδυασμού της μοντελοκεντρικής αρχιτεκτονικής με τον βασισμένου σε προοπτικές προγραμματισμό (Aspect-Oriented Programming - AOP). Η χρησιμοποίηση της τεχνικής του AOP μπορεί να καλύψει κενά και αδυναμίες της MDA. Οι μετασχηματισμοί που πραγματοποιούνται για τη δημιουργία του PSM από το PIM είναι μονοσήμαντοι και εκτελούνται κατά τη διαδικασία της μεταγλώττισης του κώδικα. Τα πιο πρόσφατα εργαλεία AOP προσφέρουν τη δυνατότητα να προστίθενται νέες δυνατότητες κατά τον χρόνο εκτέλεσης. Με αυτό τον τρόπο νέοι μετασχηματισμοί μπορούν να εισαχθούν σε προγράμματα που εκτελούνται ήδη ή κομμάτια των προγραμμάτων να αντικατασταθούν από νέες βελτιωμένες εκδόσεις.

Στην τεχνολογία του Πλέγματος ιδιαίτερα σημαντικός είναι ο μηχανισμός με τον οποίο γίνεται η υποβολή των εργασιών. Παρόλο που αυτή τη στιγμή οι διαθέσιμοι πόροι είναι περισσότεροι από αυτούς που απαιτούν οι χρήστες, στο κοντινό μέλλον η κατάσταση αυτή αναμένεται να αντιστραφεί. Ο τρόπος υποβολής των εργασιών θα πρέπει να αλλάξει και να αναζητηθούν πιο ευέλικτα σχήματα που μπορούν να προσφέρουν ποιότητα υπηρεσίας,

προτεραιότητα σε κάποιες εργασίες και γενικότερα καλύτερη διαχείριση του μηχανισμού υποβολής των εργασιών. Πιο συγκεκριμένα, το μοντέλο του μηχανισμόυ υποβολής εργασιών είναι δυνατό να χρησιμοποιεί σχεδιαστικά πρότυπα που επιτρέπουν την καλύτερη διαχείριση των εργασιών. Για παράδειγμα, με τη χρήση των προτύπων σχεδίου Command και Iterator είναι δυνατό να αναπτυχθούν μηχανισμοί ουρών προτεραιότητας για την υποβολή των εργασιών.

7 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] JINI, http://www.jini.org

[2] Project JXTA - http://www.jxta.org/

[3] Web Services Architecture, W3C (http://www.w3c.org/TR/ws-arch/)

[4] WSDL, W3C (http://www.w3c.org/TR/wsdl)

[5] SOAP, W3C (http://www.w3c.org/TR/SOAP)

[6] OASIS WSRF specifications – http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[7] OASIS Web Services Notification (WSN) TC - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[8] Web Services Addressing (WS-Addressing) –http://www.w3.org/Submission/ws-addressing/

[9] Alfonso Fuggetta, Gian Pietro Picco and Giovanni Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 342-361, May 1998.

[10] PEER-TO-PEER WORKING GROUP. 2001. Bidirectional Peer-to-Peer Communication with Interposing Fire`walls and NATs. p2pwg White Paper, Revision 0.091. May 23, 2001. http://www.peer-to-peerwg.org/tech/nat/.

[11] VEYTSEL, A. 2001. There is no P-to-P Market... But There is a Market for P-to-P. Aberdeen Group Presentation at the P2PWG, May 2001.

[12] GRAHAM, R.L. 2001. Traditional and Non-Traditional Applications; Peer-to-Peer Networks. Lecture. www.ida.liu.se/~TDTS43/tdts43-10-peer-topeer.pdf.

[13] SHIRKY, C. 2001.What is P2P... and what Isn't. An article published on O'Reilly Network. www.openp2p.com/lpt/a/p2p/2000/11/24/shirky1-whatisp2p.html.

[14] BECKER D.J., STERLING T., SAVARESE D., DORBAND J.E., RANAWAK U.A., PACKER C.V. 1995. "Beowulf: A Parallel Workstation for Scientific Computation", Proceedings of the International Conference on Parallel Processing.

[15] BARAK, A. AND LITMAN, A. 1985. MOS: a Multicomputer Distributed Operating System. Software - Practice and Experience, 15(8):725–737. August.

[16] BARAK, A. AND WHEELER, R. 1989. MOSIX: An Integrated Multiprocessor UNIX. Proceedings of the Winter 1989 USENIX Conference, pages 101–112. February.

[17] LITZKOW, M. AND SOLOMON, M. 1992. Supporting Checkpointing and Process Migration outside the UNIX Kernel. Proceedings of the USENIX Winter Conference, pages 283–290.

[18] LITZKOW, M., LIVNY, M., AND MUTKA, M. June 1988. Condor - A Hunter of Idle Workstations. Proceedings of the 8th International Conference on Distributed Computing Systems, pages 104–111

[19] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, R., WEATHERSPOON, H.,WEIMER, W.,WELLS, C., AND ZHAO, B. 2000. OceanStore: An Architecture for Global-Scale Persistent Storage. Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[20] DABEK, F., BRUNSKILL, E. KAASHOEK, F., KARGER, D., MORRIS, R., STOICA, I., AND BALAKRISHNAN, H. 2001. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII), Schloss Elmau, Germany, May 2001.

[21] WALDMAN, M., RUBIN, A. AND CRANOR, L 2000. Publius: A Robust, Tamper-Evident, Censorship-Resistant Web Publishing System. Proceedings of the USENIX Security Symposium. Denver, Colorado, USA. Aug, 2000.

[22] C. Catlett and L. Smarr, “Metacomputing,” Communications of the ACM, June 1992, pp. 44-52.

[23] FAFNER, http://www.npac.syr.edu/factoring.html

[24] Foster, J. Geisler, W. Nickless, W. Smith, S. Tuecke “Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment” in Proc. 5th IEEE Symposium on High Performance Distributed Computing. pp. 562-571, 1997.

[25] Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, International Journal of Supercomputer Applications, 11(2): 115-128, 1997.

[26] Grimshaw A., Wulf W. et al., "The Legion Vision of a Worldwide Virtual Computer". Communications of the ACM, vol. 40(1), January 1997.

[27] Distributed.Net, http://www.distributed.net/

[28] SETI@Home, http://setiathome.ssl.berkeley.edu/

[29] Ian Foster and Carl Kesselman (eds), "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, July 1998. ISBN 1-55860-475-8.

[30] David De Roure, Mark A. Baker, Nicholas R. Jennings and Nigel R. Shadbolt, "The Evolution of the Grid"

[31] OMG, http://www.omg.org

[32] Condor, http://www.cs.wisc.edu/condor/

[33] Sun Grid Engine, http://www.sun.com/software/gridware/

[34] HotPage, https://hotpage.npaci.edu/

[35] Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Nick Karonis, Matei Ripeanu, Ed Seidel, Brian Toonen. Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus. Winning Paper for Gordon Bell Prize (Special Category), Supercomputing 2001 (Revised version: August 2001)

[36] The DataGrid project, http://eu-datagrid.web.cern.ch/

[37] J. Almond and D. Snelling, "UNICORE: uniform access to supercomputing as an element of electronic commerce", Future Generation Computer Systems, 15(1999) 539-548, NH-Elsevier

[38] E. Akarsu, G.C. Fox, W. Furmanski, and T. Haupt, "WebFlow: High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", SC98: High Performance Networking and Computing, Orlando, Florida, 1998.

[39] T. Haupt, P. Bangalore, G. Henley, "Mississippi Computational Web Portal". Accepted for publication in Concurrency and Computation: Practice and Experience.

[40] Foster, C. Kesselman, S. Tuecke "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of Supercomputer Applications and High Performance Computing, 2001.

[41] NASA Information Power Grid, http://www.ipg.nasa.gov/

[42] Foster, I., Kesselman, C., Nick, J. and Tuecke, S., The Physiology of the Grid: Open Grid Services Architecture for Distributed Systems Integration, presented at GGF4, Feb. 2002 http://www.globus.org/research/papers/ogsa.pdf

[43] N. R. Jennings (2001) "An agent-based approach for building complex software systems" Comms. of the ACM, 44 (4) 35-41.

[44] M. Wooldridge and N.R. Jennings, Intelligent Agents: Theory and Practice. Knowledge Engineering Review Volume 10 No 2, June 1995.

[45] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, M. Wooldridge "Automated Negotiation: Prospects, Methods and Challenges". Journal of Group Decision and Negotiation, 10, 2, 199-215, 2001.

[46] Klaus Krauter, Rajkumar Buyya, Muthucumaru Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing"

[47] The Web Framework Map Wiki - http://www.reahl.org/wfmwiki/

[48] Apache Struts - http://struts.apache.org/

[49] Java Server Faces - http://java.sun.com/javaee/javaserverfaces/

[50] Java Foundation Classes - http://java.sun.com/products/jfc/

[51] JSR 168 - Portlet specification - http://www.jcp.org/en/jsr/detail?id=168

[52] Ruby on Rails – http://www.rubyonrails.org

[53] Microsoft ASP.NET Developer Center - http://msdn.microsoft.com/asp.net

[54] Catalyst Web Framework - http://www.catalystframework.org/

[55] CPAN - Comprehensive Perl Archive Network - http://www.cpan.org/

[56] Zope - http://www.zope.org/

[57] Dharap, M. Freeman, "Information Agents for Automated Browsing", in Proc. of the ACM CIKM'96, Rockville, USA, 1996.

[58] V. Roth, M. Jalali, R. Hartmann and C. Roland, "An Application of Mobile Agents as Personal Assistants in Electronic Commerce", in Prc. 5th Conference on the Practical Application of Intelligent Agents and Multi-Agents (PAAM'2000), Manchester, UK, April 2000.

[59] G. Neumann, "High-level Design and Architecture of an HTTP-Based Infrastructure fro Web Applications", in the World Wide Web Journal, vol. 3(1), 2000.

[60] Paolo Marques, Raul Fonseca, Paulo Simões, Luís Silva, João G. Silva, "A Component-Based Approach for Integrating Mobile Agents Into the Existing Web

Infrastructure", IEEE Symposium on Applications and the Internet (SAINT), Nara City, Nara, Japan, January 28 - February 01, 2002, pp. 100-109.

[61] Fünfrocken S., "How to Integrate Mobile Agents into Web Servers", Proc. WETICE'97 Workshop on Collaborative Agents in Distributed Web Applications, Boston, MA, June 18-20, 1997, pp. 94-99.

[62] Sun Microsystems Inc., "The Servlet Specification 2.3", http://www.javasoft.com/servlet.

[63] Sun Microsystems Inc, "JavaBeans Specification 1.01", http://www.javasoft.com/beans.

[64] Danny B. Lange, Tom Hill and Mitsuru Oshima, "A New Internet Agent Scripting Language Using XMl", AAAI-99 Workshop on AI in Electronic Commerce, Orlando, Florida, July 1999.

[65] The Apache Consortium, "The Jakarta Project", http://jakarta.apache.org/tomcat/index.html

[66] The Apache Consortium, "Apache SOAP", http://ws.apache.org/soap

[67] Iakovos Venieris, Fabrizio Zizza, and Thomas Magedanz, "Object Oriented Software Technologies in Telecommunications, From theory to Practice", John Wiley & Sons, April 2000.

[68] OMG MASIF, "Mobile Agent System Interoperability Facility (MASIF) specification", ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.

[69] IKV++ Technologies, "Grasshopper Agent platform", http://www.grasshopper.de

[70] Nikolaos Papandreou, Igor Berezovsky, Anne Lopes, Elias Eliopoulos, Jacques Chomilier: 'Universal positions in globular proteins: observation to simulation', European Journal of Biochemistry, 271:4762-4768 (2004).

[71] JBoss Portal - http://www.jboss.org/products/jbossportal

[72] Hibernate 3 - http://www.hibernate.org/

[73] MyProxy - http://grid.ncsa.uiuc.edu/myproxy/

[74] EDG Documentation - http://marianne.in2p3.fr/datagrid/documentation/

[75] gLite API - www.cern.ch/glite/

[76] BLAST - http://www.ncbi.nlm.nih.gov/blast/

[77] PDB - http://www.rcsb.org/

[78] SwissProt - http://au.expasy.org/sprot

[79] UML Specification - http://www.omg.org/cgi-bin/doc?formal/05-04-01

[80] MDA Specifications - http://www.omg.org/mda/specs.htm

[81] Alexander C., et al. (1977), A Pattern Language: Towns, Building and Construction, New York: Oxford University Press.

[82] Appleton, B. (2000), Patterns and Software: essential concepts and terminology, viewed 2 October, 2005, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html.

[83] Appleton, B. (2000), Patterns in a Nutshell, viewed 2 October, 2005, http://www.cmcrossroads.com/bradapp/docs/patterns-nutshell.html.

[84] Cunningham, W. and Back, K., (1987), 'Using Pattern Languages for Object-Oriented Programs', in the Proceedings of. OOPSLA '87, Orlando.

[85] Gamma, E., Helm, R., Johnson, R. and Vlissedes, J. (1995), Design Patterns - Elements of Reusable Object Oriented Software, Addison Wesley.

[86] Mowbray, T. and Malveau, R. (1997), CORBA Design patterns, New York: Wiley Computer Publishing.

[87] Booch, G. Object-Oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company, Inc., ISBN: 0-8053-5340-2, 1994

[88] Booch, G. and Rumbaugh, J. Unified Method for Object-Oriented Development. Documentation Set, version 0.8, technical report, Rational Software Corporation, 1995

[89] Meta-Object Facility (MOF™) - www.omg.org/mof/

[90] EMF - http://www.eclipse.org/modeling/emf/

[91] MOF QVT Final Adopted Specification - http://www.omg.org/docs/ptc/07-07-07.pdf

[92] XMI - http://www.omg.org/cgi-bin/doc?formal/2007-12-01

[93] Netbeans MDR - http://mdr.netbeans.org

[94] Java Metadata Interface - http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html

[95] Apache Velocity - http://velocity.apache.org/

[96] ArgoUML - http://argouml.tigris.org/

[97] MagicDraw - http://www.magicdraw.com/

[98] Kermeta - http://www.kermeta.org/

[99] MOFScript - http://www.modelbased.net/mofscript/index.html

[100] The IBM Model Transformation Framework (MTF) - http://www.alphaworks.ibm.com/tech/mtf

[101] The ATL Engine - http://www.sciences.univ-nantes.fr/lina/atl/

[102] ModFact - http://modfact.lip6.fr/ModFactWeb/index.jsp

[103] Kent Modelling Framework (KMF) - http://www.cs.kent.ac.uk/projects/kmf/index.html

[104] OpenArchitectureWare - http://www.openarchitectureware.org/

[105] OpenMDX - http://www.openmdx.org/index.html

[106] AndroMDA - http://www.andromda.org/

[107] XDoclet - http://www.xdoclet.org/

[108] Middlegen - http://boss.bekk.no/boss/middlegen/index.html

[109] OOMEGA - http://www.oomega.net/

[110] ArcStyler - http://www.io-software.com/

[111] MCC (Model Component Compiler) - http://www.inferdata.com/products/mcc/mdac.html

[112] OptimalJ - http://www.compuware.com/products/optimalj/default.htm

[113] Xactium XMF Mosiac - http://www.modelbased.net/www.xactium.com/

[114] SosyInc Modeler and Transformation Engine - http://www.sosyinc.com/

[115] Model-in-Action and MDA tool suite - http://www.mia-software.com/

[116] MetaEdit - http://www.metacase.com/fs.asp?paa=products.html

[117] MDWorkbench - http://www.mdworkbench.com/

[118] iQgen 3.0 - http://www.innoq.com/iqgen/

8 ΠΑΡΑΡΤΗΜΑΤΑ

8.1 ΛΙΣΤΑ ΔΗΜΟΣΙΕΥΣΕΩΝ

8.1.1 ΕΠΙΣΤΗΜΟΝΙΚΑ ΠΕΡΙΟΔΙΚΑ

1. A.I. Kostaridis, C.G. Biniaris, I.E. Foukarakis, D.I. Kaklamani, and I.S. Venieris, 'A Web-based distributed computing framework for antenna array modelling', invited paper in the special issue of the IEEE Communications Magazine on Adaptive Antennas and MIMO systems for wireless communications, October 2004, Volume: 42, Issue: 10, pp. 81 - 87.
2. I. E. Foukarakis, A. I. Kostaridis, C. G. Biniaris, D. I. Kaklamani, I. S. Venieris, 'WebMages: An agent platform based on web services'; special issue 'Emerging Middleware for Next Generation Networks' on 'Computer Communications', Volume 30, Issue 3, pp 538-545, Elsevier, February 2007.

8.1.2 ΒΙΒΛΙΑ

Ο Ι. Φουκαράκης είναι συγγραφέας δύο κεφαλαίων του επιστημονικού βιβλίου: "Advances in Information Technologies for Electromagnetics", Ed. Luciano Tarricone, των εκδόσεων Springer-Verlag:

1. D. G. Lymperopoulos, I. E. Foukarakis, A. I. Kostaridis, C. G. Biniaris, D. I. Kaklamani, "Software Agents for Parametric Computational Electromagnetics Applications" Chapter 10 in "Advances in Information Technologies for Electromagnetics", Springer-Verlag, 2007
2. I. E. Foukarakis, D. B. Logothetis, A. I. Kostaridis, D. G. Lymperopoulos, D. I. Kaklamani, "Web Services Enhanced Platform for Parametric Computational Electromagnetics Applications" Chapter 11 in "Advances in Information Technologies for Electromagnetics", Springer-Verlag, 2007

Καθώς και ενός κεφαλαίου στο "Handbook of Mobile Broadcasting: DVB-H, DMB, ISDB-T and MediaFLO", το οποίο θα εκδοθεί από τις εκδόσεις CRC Press

1. D. I. Kaklamani, I.S. Venieris, I. Foukarakis, C. Katsigiannis, “An overview of Digital Multimedia Broadcasting for Terrestrial (DMB-T)”, to appear in “Handbook of Mobile Broadcasting: DVB-H, DMB, ISDB-T and MediaFLO”, CRC Press, 2008

8.1.3 ΣΥΝΕΔΡΙΑ

1. I. E. Foukarakis, A. I. Kostaridis, C. G. Biniaris, D. I. Kaklamani and I. S. Venieris, “Implementation of a Mobile Agent Platform based on Web Services”, Proceedings of the Fifth International Workshop (Mata 2003, Marrakech, Morocco) on Mobile Agents for Telecommunication Applications, pp. 190 - 199.

2. Theodoros E. Athanaileas, Ioannis E. Foukarakis, Dimitra I. Kaklamani and Iakovos S. Venieris, ‘Using Multiple Communication Channels in a Mobile Agent Platform’, Proceedings of the Second International Workshop (Mata 2005, Montreal, Canada) on Mobility Aware Technologies And Applications, pp. 316 - 323.

3. Georgios V. Lioudakis, Ioannis E. Foukarakis, George N. Prezerakos, Dimitra I. Kaklamani, and Iakovos S. Venieris, ‘eDocuments Intelligent Enrichment from Distributed Knowledge Resources’, IEEE Eurocon 2005, Belgrade, Serbia & Montenegro, 21 - 24 November 2005

4. Kapitsaki, G.; Kateros, D.A.; Foukarakis, I.E.; Prezerakos, G. N.; Kaklamani, D.I.; Venieris, I.S., "Service Composition: State of the art and future challenges," Mobile and Wireless Communications Summit, 2007. 16th IST , vol., no., pp.1-5, 1-5 July 2007

5. Ioannis E. Foukarakis, Vassilis Atlamazoglou, Trias Thireou, Nikolaos D. Tselikas, Nikolaos. Papandreou, Dimitra I. Kaklamani, Elias Eliopoulos, “Grid Enabled Simulation of 3D Folding of Proteins By Predicting Critical Amino Acid Positions”, 1st HellasGrid User Forum, Athens, Greece, 10 – 11 January 2008.

8.1.4 ΔΗΜΟΣΙΕΥΣΕΙΣ ΣΕ ΔΙΑΔΙΚΑΣΙΑ ΚΡΙΣΗΣ

1. D. A. Kateros, D. A. Zarbouti, D. C. Tsilimantos, C. I. Katsigiannis, P. K. Gkonis, I. E. Foukarakis, D. I. Kaklamani and I. S. Venieris, "DVB-T network planning: A case study for Greece", submitted on IEEE Antennas and Propagation Magazine.
2. Ioannis E. Foukarakis, Vassilis Atlamazoglou, Trias Thireou, Nikolaos D. Tselikas, Nikolaos. Papandreou, Dimitra I. Kaklamani, Elias Eliopoulos, “mirGRID: a web-server for the determination of the Most Interacting Residues that participate in the formation and stabilization of a protein globule”, submitted on Nucleic Acids Research.

8.1.5 ΑΝΑΦΟΡΕΣ

A.I. Kostaridis, C.G. Biniaris, I.E. Foukarakis, D.I. Kaklamani, and I.S. Venieris, ‘A Web-based distributed computing framework for antenna array modelling’, invited paper in the special issue of the IEEE Communications Magazine on Adaptive Antennas and MIMO systems for wireless communications, October 2004, Volume: 42, Issue: 10, pp. 81 – 87, cited by:

1. Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, ‘Toward Synthesis of Web Services and Mobile Agents’, in proceedings of Workshop on Web Services and Agent-based Engineering 2004 (WSABE2004)

2. Risa Tanaemi, Ryuichi Saita, Kazuto Tominaga, ‘Constructing a mobile agent system coping with obstructions to agent migration caused by firewalls and offline hosts’ in proceedings of the 6th JSSST Workshop on Internet Technology (WIT2004)

3. Hoon Kang Neo, Kim Meow Liew, QingPing Lin, ‘A Grid-Based Mobile Agent Collaborative Virtual Environment’ in proceedings of 2005 International Conference on Cyberworlds (CW'05) pp. 335-339

4. N. Blefari-Melazzi, G. Ceneri, G. Cortese, N. Davies, N. Dellas, A. Friday, J. Hamard, E. Koutsoloukas, C. Niedermeier, C. Noda, J. Papanis, C. Petrioli, E. Rukzio, O. Storz, J. Urban: "The Simplicity Project: easing the burden of using complex and heterogeneous ICT devices and services. Part II: State of the Art of

Related Technologies", IST Mobile&Wireless Communications Summit 2004, June 27-30 2004, Lyon, France.

8.2 TEMPLATES

8.2.1 POJO

pojo.vm

```
import java.util.Date;
/**
 *
 * Java Bean to hold information about $pojo.getName()
 *
 * @author $author
 *
 */

 public class $pojo.getName() {

     protected Long key = null;
     #foreach($field in $tool.filterAttributes($pojo))
     protected $field.getType().getName() $field.getName();
     #end

     public $pojo.getName()() {

     }

     public Long getKey() {
          return key;
     }

     public void setKey(Long key) {
          this.key = key;
     }

     #foreach($field in $tool.filterAttributes($pojo))
     public $field.getType().getName()
get$tool.capitalizeFirstLetter($field.getName())() {
          return this.$field.getName();
     }
```

```
     public void
set$tool.capitalizeFirstLetter($field.getName())($field.getType().getName(
) val) {
          this.$field.getName() = val;
     }
     #end
 }
```

8.2.2 CRUD ΓΙΑ POJOS

service.vm

```
import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.criterion.Expression;

import java.util.List;

public class $pojo.getName()Service {
      private final static $pojo.getName()Service singleton = new
$pojo.getName()Service();
      private SessionFactory sessionFactory = null;

      private $pojo.getName()Service() {
            init();
      }

      public static $pojo.getName()Service getInstance() {
            return singleton;
      }

      protected void init() {
            Configuration config = new Configuration();
            config.configure("hibernate.cfg.xml");
            this.sessionFactory = config.buildSessionFactory();
      }

      protected void addNew$pojo.getName()($pojo.getName() item) {
            if(sessionFactory == null)
```

```
                init();
        Session session = this.sessionFactory.openSession();
        Transaction t = session.beginTransaction();
        try {
                session.save(item);
                t.commit();
        } catch (Exception e) {
                t.rollback();
        } finally {
                if (session != null) {
                        session.disconnect();
                }
        }
}

protected List findAll() {
        if(sessionFactory == null)
                init();
        Session session = this.sessionFactory.openSession();
        Criteria criteria = session
                        .createCriteria(${pojo.getName()}.class);
        List results = criteria.list();
        session.disconnect();
        return results;
}

protected $pojo.getName() find$pojo.getName()(String key) {
        if(sessionFactory == null)
                init();
        Session session = this.sessionFactory.openSession();
        Criteria criteria = session
                        .createCriteria(${pojo.getName()}.class);
                        List results = criteria.list();
        session.disconnect();
                        if(results.size() == 1)
                return ($pojo.getName()) results.get(0);
        return null;
}

protected void update$pojo.getName()($pojo.getName() item) {
        if(sessionFactory == null)
```

```
                  init();
            Session session = this.sessionFactory.openSession();
            Transaction t = session.beginTransaction();
            try {
                  session.update(item);
                  t.commit();
            } catch (Exception e) {
                  t.rollback();
            } finally {
                  if (session != null) {
                        session.disconnect();
                  }
            }
      }

      protected void delete$pojo.getName()(String key) {
            if(sessionFactory == null)
                  init();
            Session session = this.sessionFactory.openSession();
            Transaction t = session.beginTransaction();
            try {
                  $pojo.getName() item = ($pojo.getName()) session
                              .load(${pojo.getName()}.class,
                                          new Long(key));

                  session.delete(item);
                  t.commit();
            } catch (Exception e) {
                  t.rollback();
            } finally {
                  if (session != null) {
                        session.disconnect();
                  }
            }
      }
}
```

create.vm

```
<form method="POST" action="/Controller">
  <table border="0" width="100%">
#foreach($field in $tool.filterAttributes($pojo))
```

```
    <tr>
      <td>$field.getName()</td>
      <td>
        <jsp:include page="types/${field.getType().getName()}.jsp" >
          <jsp:param name="attrName" value="$field.getName()" />
        </jsp:include>
      </td>
    </tr>
#end
  </table>
  <input type="hidden" name="pojo_type" value="$pojo.getName()" />
  <input type="submit" value="Create" name="op" />
</form>
```

retrieve.vm

```
<form method="POST" action="/Controller">
  <table border="0" width="100%">
#foreach($field in $tool.filterAttributes($pojo))
    <tr>
      <td>$field.getName()</td>
      <td>
        <jsp:include page="types/${field.getType().getName()}.jsp" >
          <jsp:param name="attrName" value="$field.getName()" />
        </jsp:include>
      </td>
    </tr>
#end
  </table>
</form>
```

update.vm

```
<form method="POST" action="/Controller">
  <table border="0" width="100%">
<%
$pojo.getName() item = ($pojo.getName())
request.getParameter("$pojo.getName()_item");
%>
#foreach($field in $tool.filterAttributes($pojo))
    <tr>
      <td>$field.getName()</td>
      <td>
```

```
        <jsp:include page="types/${field.getType().getName()}.jsp" >
          <jsp:param name="attrName" value="$field.getName()" />
          <jsp:param name="attrValue"
value="<%=item.get$tool.capitalizeFirstLetter($field.getName())()%>" />
        </jsp:include>
      </td>
    </tr>
#end
  </table>
  <input type="hidden" name="pojo_type" value="$pojo.getName()" />
  <input type="submit" value="Update" name="op" />
</form>
```

delete.vm

```
<form method="POST" action="/Controller">
  <table border="0" width="100%">
    <tr>
#foreach($field in $tool.filterAttributes($pojo))
      <th>$field.getName()</th>
#end
      <th>Delete</th>
    </tr>
<%
                        List items = (List)
request.getParameter("$pojo.getName()_list");
                        Iterator iter = items.iterator();
                        while(iter.hasNext())
                        $pojo.getName() item = ($pojo.getName())
iter.next();
%>
    <tr>
#foreach($field in $tool.filterAttributes($pojo))

<td><%=item.get$tool.capitalizeFirstLetter($field.getName())()%></td>
#end
      <td><input type="checkbox" name="item_<%=item.getKey()%>"
value="<%=item.getKey()%>" /></td>
    </tr>
<%
}
%>
```

```
  </table>
  <input type="hidden" name="pojo_type" value="$pojo.getName()" />
  <input type="submit" name="op" value="Delete selected" />
</form>
```

8.2.3 ΡΥΘΜΙΣΕΙΣ HIBERNATE

hibernate-mapping.vm

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class
      name="$pojo.getName()"
      table="$pojo.getName()">
      <cache usage="read-write"/>
      <id
         name="key"
         column="uid"
         type="java.lang.Long">
         <generator class="native">
            <param name="sequence">$pojo.getName()_seq</param>
         </generator>
      </id>
#foreach($field in $pojo.getFeature())
      <property
         name="$field.getName()"
         type="$tool.getHibernateType($field.getType().getName())"
         update="false"
         insert="true"
         column="$field.getName()"
         unique="false"/>
#end
   </class>
</hibernate-mapping>
```

8.2.4 ABSTRACT FACTORY

Factory.vm

```
import java.util.Hashtable;

/**
 *
 * Factory interface for $c.getName()
 *
 * @author $author
 *
 */

 public class $c.getName()Factory {

      private static Hashtable factories = new Hashtable();
      private static I$c.getName()Factory defaultFactory = null;

     public static void registerFactory(String name, I$c.getName()Factory
factory) {
        factories.put(name, factory);
      }

      public static I$c.getName()Factory getFactory(String name) {
              return (I$c.getName()Factory) factories.get(name);
      }

      public static I$c.getName()Factory getDefaultFactory() {
              return defaultFactory;
      }

 }
```

factoryimpl.vm

```
import java.util.*;

/**
 *
 * Local proxy for $c.getName()
 *
 * @author $author
 *
 */
```

```
 public class $c.getName()FactoryImpl implements
I$factoryClass.getName()Factory {
#foreach($field in $tool.filterAttributes($pojo))
     protected $field.getType().getName() $field.getName();
#end

     public $c.getName()FactoryImpl () {
            $factoryClass.getName().registerFactory($c.getName(), this);
     }

#foreach($item in $factoryMethods)
     public $item.getType().getName() $field.getName()(#foreach($param in
$tool.getParameters($item)) $sep$param.getType().getName()
$param.getName() #set ( $sep = ', ' ) #end) {
     }
#end
 }
```

8.2.5 SINGLETON

singleton.vm

```
import java.util.Date;
/**
 *
 * Singleton class $c.getName()
 *
 *
 */

 public class $c.getName() {

#foreach($field in $tool.filterAttributes($c))
     protected $field.getType().getName() $field.getName();
#end

     private static final $c.getName() INSTANCE = new $c.getName()();

     private $c.getName()() {

     }
```

```
     public synchronized static $c.getName() getInstance() {
          return INSTANCE;
     }

#foreach($field in $tool.filterAttributes($c))
     public $field.getType().getName()
get$tool.capitalizeFirstLetter($field.getName())() {
          return this.$field.getName();
     }

     public void
set$tool.capitalizeFirstLetter($field.getName())($field.getType().getName(
) val) {
          this.$field.getName() = val;
     }
#end

#foreach($field in $tool.filterOperations($c))
#set( $rparam = $tool.getReturnParameter($field))
#if($tool.getReturnParameter($field))
#set( $sep = '' )
     $tool.getVisibility($field) $rparam.getType().getName()
$field.getName()(#foreach($param in $tool.getParameters($field))
$sep$param.getType().getName() $param.getName() #set ( $sep = ', ' ) #end)
{
          //TO DO: implement logic

     }
#else
#set( $sep = '' )
     $tool.getVisibility($field) void $field.getName()(#foreach($param in
$tool.getParameters($field)) $sep$param.getType().getName()
$param.getName() #set ( $sep = ', ' ) #end) {
          //TO DO: implement logic

     }
#end
#end
}
```

controller.vm

```
import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.http.*;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;

public class $pojo.getName()Controller extends HttpServlet {

      protected void doPost(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
IOException {
            doGet(request, response);
      }



      protected void doGet(HttpServletRequest request,
                  HttpServletResponse response) throws ServletException,
IOException {

            String op = request.getParameter("op");
            if(op.equalsIgnoreCase("Create")) {

                  $pojo.getName() item = new $pojo.getName()();

#foreach($field in $tool.filterAttributes($pojo))
                  String ${field.getName()} =
request.getParameter("$field.getName()");

      item.set$tool.capitalizeFirstLetter(${field.getName()})(new
$field.getType().getName()(${field.getName()}));
#end
```

```
      ${pojo.getName()}Service.getInstance().addNew${pojo.getName()}(item)
;
                  RequestDispatcher disp;
                  disp =
getServletContext().getRequestDispatcher("${pojo.getName()}_retrieve.jsp")
;
                  request.setAttribute("item", item);
                  disp.forward(request, response);
            }
            if(op.equalsIgnoreCase("Retrieve")) {
                  String key = request.getParameter("key");
                  ${pojo.getName()} item =
${pojo.getName()}Service.getInstance().find${pojo.getName()}(key);
                  request.setAttribute("item", item);
                  RequestDispatcher disp;
                  disp =
getServletContext().getRequestDispatcher("$pojo.getName()_retrieve.jsp");
                  disp.forward(request, response);
            }
            if(op.equalsIgnoreCase("Update")) {

                  $pojo.getName() item = new ${pojo.getName()}();
#foreach($field in $tool.filterAttributes($pojo))
                  String ${field.getName()} =
request.getParameter("$field.getName()");

      item.set$tool.capitalizeFirstLetter(${field.getName()})(new
$field.getType().getName()(${field.getName()}));
#end

      ${pojo.getName()}Service.getInstance().update${pojo.getName()}(item)
;
                  RequestDispatcher disp;
                  disp =
getServletContext().getRequestDispatcher("${pojo.getName()}_retrieve.jsp")
;
                  request.setAttribute("item", item);
                  disp.forward(request, response);
            }
            if(op.equalsIgnoreCase("Delete selected")) {
```

```
                  Enumeration en = request.getParameterNames();
                  while(en.hasMoreElements()) {
                        String name = (String) en.nextElement();
                        if(name.startsWith("item_")) {
                              String[] key =
request.getParameterValues(name);
                              if((key != null) && (key.length > 0))

      ${pojo.getName()}Service.getInstance().delete${pojo.getName()}(key[0
]);
                        }
                  }
                  request.setAttribute("items",
${pojo.getName()}Service.getInstance().findAll());
                  RequestDispatcher disp;
                  disp =
getServletContext().getRequestDispatcher("$pojo.getName()_delete.jsp");
                  disp.forward(request, response);
            }

            RequestDispatcher disp;
            disp = getServletContext().getRequestDispatcher("index.jsp");
            disp.forward(request, response);
      }
}
```

8.2.6 PROXY

local.vm

```
import java.util.Date;

/**
 *
 * Local proxy for $c.getName()
 *
 * @author $author
 *
 */

 public class $c.getName()Proxy {
```

```
                            private $c.getName() obj;

     public $c.getName()(${c.getName()} item) {
       this.obj = item;
     }

     #foreach($field in $tool.filterProxyOperations($c))
     public $field.getType().getName() $field.getName()(#foreach($param in
$tool.getParameters($field)) $sep$param.getType().getName()
$param.getName() #set ( $sep = ', ' ) #end) {
          obj.$field.getName()(#foreach($param in
$tool.getParameters($field)) $sep $param.getName() #set ( $sep = ', ' )
#end);
     }
     #end
 }
```

8.2.7 ADAPTER

8.2.8 ITERATOR

8.2.9 COMMAND

command.vm

```
import java.util.Date;

/**
 *
 * Command for $f.getName().
 *
 * @author $author
 *
 */

 public class $tool.capitalizeFirstLetter($f)Command implements ICommand {

                         #foreach($param in $tool.getParameters($f))
                             private $param.getType().getName()
$param.getName();
                         #end
```

```
                            private $c.getName() obj;

     public $f.getName()Command($c.getName() item) {
       this.obj = item;
     }

     public void execute() {
                        obj.${f.getName()}(#foreach($param in
$tool.getParameters($f)) $sep this.$param.getName() #set ( $sep = ', ' )
#end);
     }

     #foreach($field in $tool.filterProxyOperations($c))
     public $field.getType().getName() $field.getName()(#foreach($param in
$tool.getParameters($field)) $sep$param.getType().getName()
$param.getName() #set ( $sep = ', ' ) #end) {
          obj.$field.getName()(#foreach($param in
$tool.getParameters($field)) $sep $param.getName() #set ( $sep = ', ' )
#end);
     }
     #end
 }
```

8.2.10 OBSERVER

observer.vm

```
import java.util.Date;
import java.util.Observable;          //Observable is here

/**
 *
 * Observable Java Bean $c.getName()
 *
 * @author $author
 *
 */

 public class $c.getName() extends Observable {

     protected Long key = null;
```

```
     #foreach($field in $tool.filterAttributes($c))
     protected $field.getType().getName() $field.getName();
     #end

     public $c.getName()() {

     }

     public void onChangeNotify(String message) {
        notifyObservers(message);
      }

     #foreach($field in $tool.filterAttributes($c))
     public $field.getType().getName()
get$tool.capitalizeFirstLetter($field.getName())() {
          return this.$field.getName();
     }

     public void
set$tool.capitalizeFirstLetter($field.getName())($field.getType().getName(
) val) {
          this.$field.getName() = val;
     }
     #end
 }
```

8.3 ΚΙΝΗΤΟΙ ΠΡΑΚΤΟΡΕΣ

stationary.vm

```
import gr.ntua.ifouk.agent.StationaryAgent;
import gr.ntua.ifouk.agent.ThreadInterrupt;
import gr.ntua.ifouk.agent.rpc.CallResponse;
import gr.ntua.ifouk.agent.rpc.SynchCall;
import gr.ntua.ifouk.agent.helpers.*;

import java.util.*;

public class $c.getName() extends StationaryAgent {

      //States
#foreach($item in $states)
```

```
      private static final String $item.getName().toUppercase() =
"$item.getName()";
#end
      //Current state
      private String currentState = $start.getName();

#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isSingleton($citem) || $tool.isPoolable($citem))
      private transient $citem.getName() $citem.getName().toLowercase();
#end
#end

      public void destroy() {
            // No need to release transient - only pooled items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))

$citem.getName()Pool.getInstance().release($citem.getName().toLowercase())
;
#end
#end
      }

      public void init() {
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isSingleton($citem) || $tool.isPool($item))
            // No need to release transient - only pooled items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))
             $citem.getName().toLowercase() =
$citem.getName()Pool.getInstance().acquire();
#end
#if($tool.isSingleton($citem))
            $citem.getName().toLowercase() =
$citem.getName().getInstance();
#end
#end
#end

      }
```

```
      public void start() throws ThreadInterrupt {

#foreach($item in $states)
          if(currentState.compareTo("$item.getName()") == 0) {
#set($op = $item.getEntry().getOperation())
#set($sep = '')
                  ${op.getName()}(#foreach($ar in
$item.getEntry().getActualArgument() $sep $ar.getValue().getBody()  #set (
$sep = ', ' ) #end);
      }
#end

      }

}
```

mobile.vm

```
import gr.ntua.ifouk.agent.StationaryAgent;
import gr.ntua.ifouk.agent.ThreadInterrupt;
import gr.ntua.ifouk.agent.rpc.CallResponse;
import gr.ntua.ifouk.agent.rpc.SynchCall;
import gr.ntua.ifouk.agent.helpers.*;

import java.util.*;

public class $c.getName() extends MobileAgent {

      //States
#foreach($item in $states)
      private static final String $item.getName().toUppercase() =
"$item.getName()";
#end
      //Current state
      private String currentState = $start.getName();

#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isSingleton($citem) || $tool.isPoolable($citem))
      private transient $citem.getName() $citem.getName().toLowercase();
#end
#end
```

```
      public void destroy() {
            // No need to release transient - only pooled items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))

$citem.getName()Pool.getInstance().release($citem.getName().toLowercase())
;
#end
#end
      }

      public void init() {
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isSingleton($citem) || $tool.isPool($item))
            // No need to release transient - only pooled items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))
             $citem.getName().toLowercase() =
$citem.getName()Pool.getInstance().acquire();
#end
#if($tool.isSingleton($citem))
            $citem.getName().toLowercase() =
$citem.getName().getInstance();
#end
#end
#end

      }

      public void start() throws ThreadInterrupt {

#foreach($item in $states)
          if(currentState.compareTo("$item.getName()") == 0) {
#set($op = $item.getEntry().getOperation())
#set($sep = '')
                  ${op.getName()}(#foreach($ar in
$item.getEntry().getActualArgument() $sep $ar.getValue().getBody()  #set (
$sep = ', ' ) #end);
      }
#end
```

```
      }


      public void afterMove() {
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isSingleton($citem) || $tool.isPool($item))
            // No need to release transient - only pooled items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))
             $citem.getName().toLowercase() =
$citem.getName()Pool.getInstance().acquire();
#end
#if($tool.isSingleton($citem))
            $citem.getName().toLowercase() =
$citem.getName().getInstance();
#end
#end
#end
      }

      public void beforeMove() {
                        // No need to release transient - only pooled
items
#foreach($citem in $tool.getAssociatedClasses($c))
#if($tool.isPoolable($citem))

$citem.getName()Pool.getInstance().release($citem.getName().toLowercase())
;
#end
#end
      }

}
```

8.4 JDL

simplejdl.vm

```
Executable = "$command.getExecutable()";
StdOutput = "stdout.txt";
StdError = "stderr.txt";
```

```
VirtualOrganisation = "see";
#set($sep = '')
InputSandbox = {#foreach($item in $command.getInput())
$sep"$item.getName()" #set ($sep = ', ') #end};
OutputSandbox = {#foreach($item in $command.getOutput())
$sep"$item.getName()" #set($sep = ', ') #end};
Requirements = (other.GlueCEStateStatus == "Production");
Rank = other.GlueCEStateFreeCPUs;
```

parametric.jdl

```
JobType = "Parametric";
Executable = "$command.getExecutable()";
Parameters = $command.getParameters().size();
ParameterStart = $command.getParameterStart();
ParameterStep = $command.getParameterStep();
Arguments = "_PARAM_";
StdOutput = "parametric_PARAM_.out";
StdError = "parametric_PARAM_.err";
InputSandbox = {#foreach($item in $command.getInput())
$sep"$item.getName()" #set ($sep = ', ') #end};
OutputSandbox = {#foreach($item in $command.getOutput())
$sep"$item.getName()_PARAM_" #set($sep = ', ') #end};
Requirements = (other.GlueCEStateStatus == "Production");
Rank = other.GlueCEStateFreeCPUs;
```